

AD-A160 846

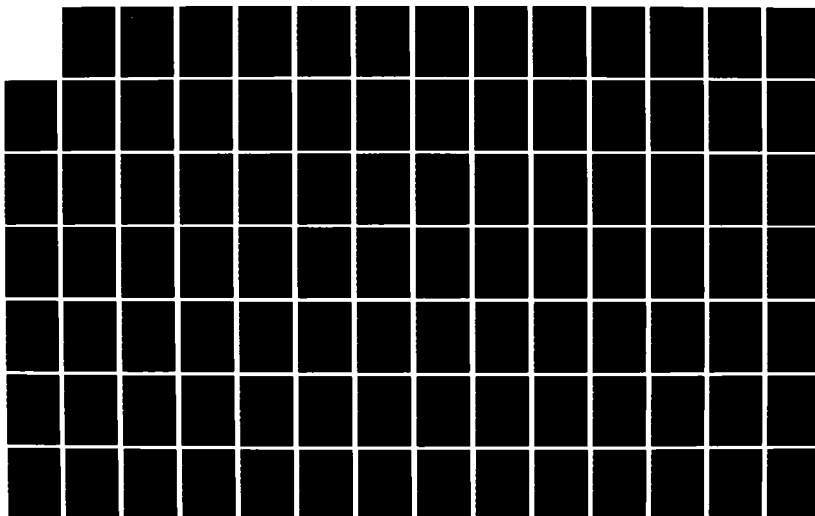
SCAN LINE DIFFERENCE COMPRESSION ALGORITHM SIMULATION
STUDY(U) DELTA INFORMATION SYSTEMS INC MORSHAM PA
AUG 85 NCS-TIB-85-6 DCA100-83-C-0047

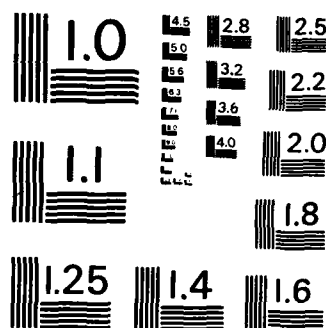
1/2

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

NCS TIB 85-6

AD-A160 846



NATIONAL COMMUNICATIONS SYSTEM

TECHNICAL INFORMATION BULLETIN

85-6

SCAN LINE DIFFERENCE COMPRESSION ALGORITHM SIMULATION STUDY

AUGUST 1985

DTIC FILE COPY

DTIC



75

85 11 01 037

NCS TECHNICAL INFORMATION BULLETIN 85-6

SCAN LINE DIFFERENCE COMPRESSION
ALGORITHM SIMULATION STUDY

AUGUST 1985

PROJECT OFFICER

EDWARD P. GREENE
Senior Electronics Engineer
Office of NCS Technology
and Standards

APPROVED FOR PUBLICATION:

Dennis Bealson
for MARSHALL L. CAIN
Assistant Manager
Office of NCS Technology
and Standards

FOREWORD

Among the responsibilities assigned to the Office of the Manager, National Communications System, is the management of the Federal Telecommunication Standards Program. Under this program, the NCS, with the assistance of the Federal Telecommunication Standards Committee identifies, develops, and coordinates proposed Federal Standards which either contribute to the interoperability of functionally similar Federal telecommunication systems or to the achievement of a compatible and efficient interface between computer and telecommunication systems. In developing and coordinating these standards, a considerable amount of effort is expended in initiating and pursuing joint standards development efforts with appropriate technical committees of the Electronic Industries Association, the American National Standards Institute, the International Organization for Standardization, and the International Telegraph and Telephone Consultative Committee of the International Telecommunication Union. This Technical Information Bulletin presents an overview of an effort which is contributing to the development of compatible Federal, national, and international standards in the area of facsimile standards. It has been prepared to inform interested Federal activities of the progress of these efforts. Any comments, inputs or statements of requirements which could assist in the advancement of this work are welcome and should be addressed to:

Office of the Manager
National Communications System
ATTN: NCS-TS
Washington, DC 20305
(202) 692-2124

This document has been approved
for publication and sale; its
distribution is unlimited.



SCAN LINE DIFFERENCE COMPRESSION
ALGORITHM SIMULATION STUDY

August, 1985

Accession For	
NTIS	<input checked="checked" type="checkbox"/>
DTIC	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
Fv	
Distribution/	
Availability Codes	
Dist	Special
A-1	

Final Report

Submitted to:

NATIONAL COMMUNICATIONS SYSTEM
Office of Technology and Standards
Washington, DC 20305

Contracting Agency:

DEFENSE COMMUNICATIONS AGENCY

Contract Number - DCA100-83-C-0047

DELTA INFORMATION SYSTEMS, INC.

Horsham Business Center, Bldg. 3

300 Welsh Road

Horsham PA 19044

Table of Contents

<u>Section</u>	<u>Page</u>
1.0 Introduction.	1-1
2.0 Conditioned Image File Generation and Analysis.	2-1
3.0 SLDC Encoder Software Design.	3-1
3.1 Functional Description.	3-1
3.2 Software Documentation.	3-5
4.0 SLDC Decoder Software Design.	4-1
4.1 Functional Description.	4-1
4.2 Software Documentation.	4-2
5.0 SLDC Compression Results and Analysis	5-1
6.0 Conclusions and Recommendations	6-1
 APPENDIX A - DESCRIPTION OF SLDC ALGORITHM	
 APPENDIX B - SOFTWARE OPERATING INSTRUCTIONS	
 APPENDIX C - CODE LISTINGS	

1.0 Introduction

This document summarizes work performed by Delta Information Systems, Inc. (DIS), for the National Communications System, an organization of the U. S. Government, under Task 2.0 of the Modification P00007 of Contract DCA100-83-C-0047. The purpose of this task was to investigate the efficiency and potential operational effectiveness of the Scan Line Difference Compression (SLDC) algorithm presented in Appendix A. Under this task, DIS performed a software simulation study of the SLDC algorithm. The software was run using selected CCITT binary standard images as input, and the results were compared with those of the MODREAD II compression algorithm run with the same input data. The MODREAD II algorithm was chosen for the comparison because it is the most effective compression technique currently available.

DIS has analyzed the SLDC algorithm and has generated tapes containing the Conditioned Image Files (CIF's) for the following twelve (12) combinations of the binary images and resolutions that were processed:

<u>Image</u>	<u>Resolution</u>
CCITT Image #1	200 lines/inch
CCITT Image #5	240 lines/inch
CCITT Image #7	300 lines/inch
	400 lines/inch

Each Subtask of Task 2.0 is presented in a section of this

report. Section 2.0 discusses the first subtask, in which Delta Information Systems studied a number of scan line conditioning techniques and generated the CIF's and corresponding prediction statistics using the technique selected in the study. In Section 3.0, the second subtask is presented, in which the FORTRAN software modules associated with the implementation the SLDC encoding algorithm are described both narratively and with structure charts, data flow charts, and software specifications.

Section 4.0 includes the third subtask, in which the software modules associated with the SLDC decoding algorithm are described; again, the software is described both narratively and with structure charts, data flow charts, and software specifications. The fourth subtask, in which Delta Information Systems ran the SLDC encoding program on each image with various sets of design parameters, is presented in Section 5.0. The compression statistics for all 168 simulation runs performed by Delta Information Systems are presented in this section.

2.0 Conditioned Image File Generation and Analysis

The objective of line conditioning is to increase the compressibility of the image under the constraint that the original image can be reconstructed from the CIF without distortion. Delta Information Systems analyzed a number of scan line conditioning techniques in this subtask and generated twelve Conditioned Image Files (CIF's), one for each image/resolution combination, with the technique selected. An example of a CIF is presented in Figure 2.1; the test image from which it was generated appears in Figure 2.2.

The conditioning technique selected by DIS predicts the binary state of an image element based on a weighted average of four of its close neighbors and produces a file of predictions, where correct predictions are '0's and incorrect predictions are '1's. Because each predicted element is based on the state of four neighboring elements (see Section A-2.1 in Appendix A), the predictor conditioning algorithm is completely described by a sixteen-entry state table; the state table employed in this study is presented in Table 2.1.

It was originally anticipated that each CCITT image would produce a different state table; however, this was not the case. DIS generated CIF prediction statistics for each file of the input image set. The results that were tabulated included the frequency of occurrence of each state together with the frequency of occurrence of a '0' pel for that state; this data revealed that all of the input images produced the same predictor conditioning algorithm, the one represented by the state table in Table 2.1.

Figure 2.1 - Conditioned Image File - Kanji 2001.jp1

THE UNIVERSITY OF CHICAGO








Figure 1. The effect of the concentration of the inhibitor on the rate of polymerization of α -methylstyrene in the presence of SnCl_4 at 25°C .

金瓶梅詞話卷之五

சென்னை, 19.12.2019

100-44388-100

齊

三

Figure 1. The effect of the concentration of the inhibitor on the rate of polymerization of α -methylstyrene in the presence of SnCl_4 at 25°C .

1770-1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 1783, 1784, 1785, 1786, 1787, 1788, 1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1798, 1799, 1800, 1801, 1802, 1803, 1804, 1805, 1806, 1807, 1808, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1840, 1841, 1842, 1843, 1844, 1845, 1846, 1847, 1848, 1849, 1850, 1851, 1852, 1853, 1854, 1855, 1856, 1857, 1858, 1859, 1860, 1861, 1862, 1863, 1864, 1865, 1866, 1867, 1868, 1869, 1870, 1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 245

3

[illegible]

大德

Table 2.1 - Predictor Conditioning Algorithm State Table

<u>A</u> <u>B</u> <u>C</u> <u>D</u>	<u>PREDICTION</u>
0 0 0 0	0
0 0 0 1	1
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	1
1 0 0 0	0
1 0 0 1	0
1 0 1 0	0
1 0 1 1	1
1 1 0 0	0
1 1 0 1	1
1 1 1 0	0
1 1 1 1	1

DIS generated run length statistics using the selected predictor conditioning algorithm. For each CIF, a set of statistics was tabulated; the statistics generated include the following:

1. Number of bits in CIF;
2. Weight of CIF;
3. Histogram of the weight of the Conditioned Scan Lines (CSL's) within the CIF;
4. Histogram of run lengths in the CIF; and,
5. Two histograms of the weight of CIF segments, where each CIF segment represents a contiguous 32-bit or 64-bit section of the CSL.

Delta Information Systems generated a Conditioned Image File for each test image at each resolution in the input set. The set of twelve CIF's was written onto a 9-track, 1600 BPI magnetic tape as described in the SOW; DIS will deliver two copies of this tape. The operating instructions for the software modules employed to generate the CIF's and their associated statistics are included in Appendix B; the code listings for these modules appear in Appendix C.

3.0 SLDC Encoder Software Design

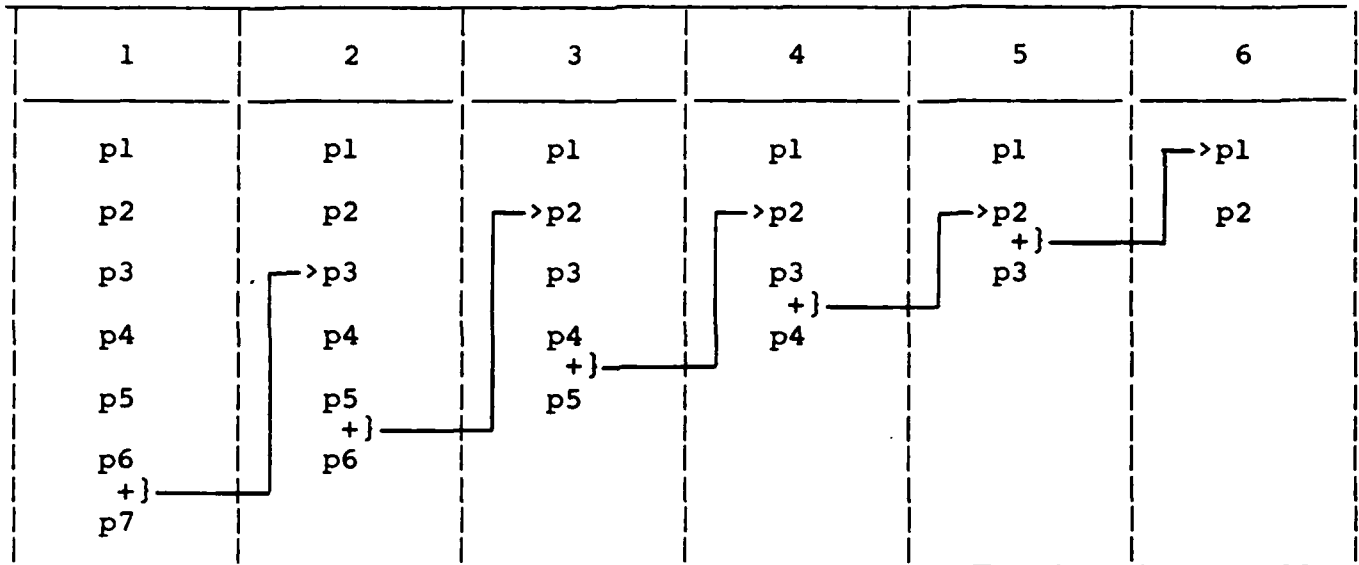
Scan Line Difference Compression is an encoding method employed in the transmission of binary images. The SLDC encoder program compresses the Conditioned Image Files (CIF's) generated in the previous subtask by taking advantage of the local conditional statistics of each image. The SLDC algorithm is described in detail in Appendix A; a brief description of the encoding process is presented here, along with the documentation for the software modules associated with the encoder program. The operating instructions for the encoder program are presented in Appendix B; the code listings appear in Appendix C.

3.1 Functional Description

File encoding is done one line at a time. As each Conditioned Scan Line (CSL) is read in, the Hamming weight of the line is calculated. If the line weight is zero, the line is encoded with a single bit, set to one (1). If the line weight is greater than zero, the integer value of the line weight is placed in the first thirteen bits of the Encoded Scan Line (ESL) (the integer value of the line weight does not exceed 2^{12}) and the CSL is encoded as a series of n-length segments, where n is the user-defined segment length. The codewords for all possible segment weights which can occur on a line of the current line weight are determined from the statistics of the line; a Huffman coding technique, illustrated in Figure 3.1, is employed to generate the codewords.

The next step is to partition the CSL into segments of the length

Huffman Probability Table Reduction Process



Huffman Codeword Table Expansion Process

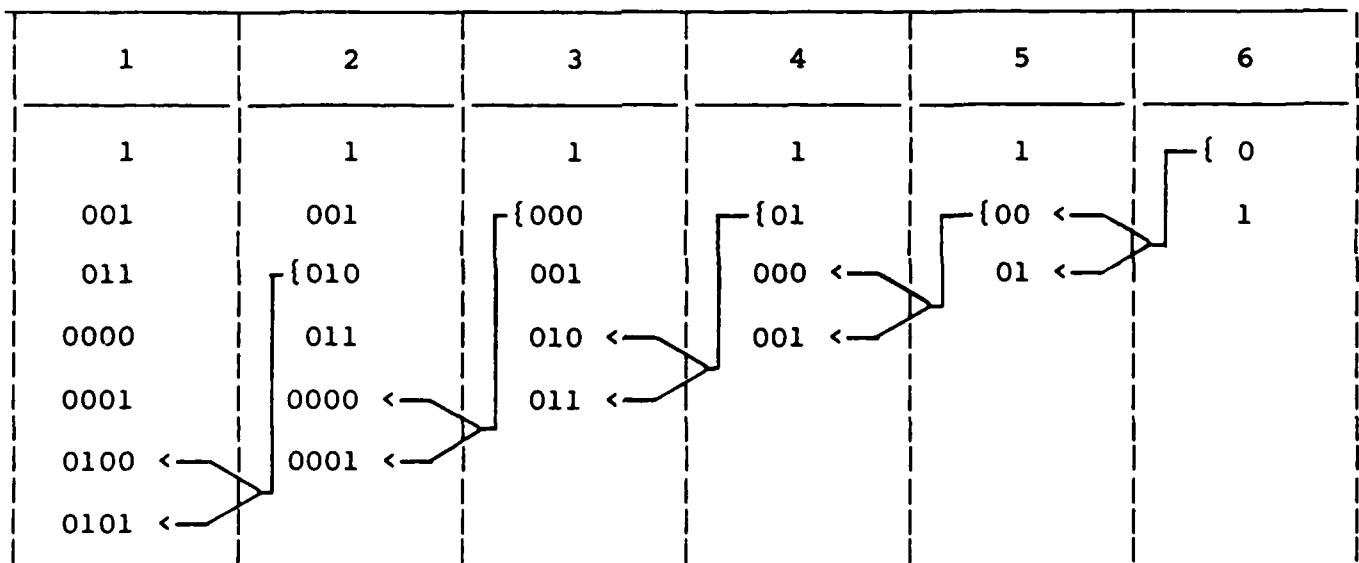


Figure 3.1 - Huffman Codeword Assignment Process

specified by the operator. Each segment is encoded with a two-part Segment State Word (SSW) that consists of a segment weight codeword and a segment rank. As each segment is extracted from the CSL, it is placed into a buffer and its weight is calculated. If the segment weight is zero, the zero-length codeword is placed into the SSW and the segment rank, which is an integer value which indicates the positions of the ones (1's) in the segment, is omitted.

If the segment weight is greater than zero and less than the maximum weight allowable for a segment of that size, the rank encoding procedure is initiated. The segment buffer is examined from left to right to determine the rank of the segment using the method described in Appendix A. Once the rank is determined, it is placed, along with the segment weight codeword, in the output buffer using the prescribed number of bits generated by the rank length equation.

If the segment weight is greater than the maximum allowable weight, the rank encoding procedure is not invoked; the SSW for the segment is comprised of the codeword for the "exceeded maximum" condition followed by the segment in uncompressed form. Maximum segment weight values must be employed in order to avoid arithmetic overflow in the rank encoding computations. The maximum segment weights for the various segment lengths are presented in Table 3.1; these values are slightly different from those found in Appendix A because the HP-1000 simulation system employs a sign-bit in its 32-bit double integer word. Therefore, the HP-1000 has a maximum integer value of 2^{31} instead of the value of 2^{32} apparently employed to construct Table A-1.

Segment Length in Bits	Maximum Segment Weight Value	
	16-bit Word Length	32-bit Word Length
<18	unconstrained	unconstrained
20	6	"
24	5	"
28	4	"
32	4	"
36	4	12
40	3	10
44	3	9
48	3	9
52	3	8
56	3	8
60	3	7
64	3	7

Table 3.1 - Maximum Weight Segment for Normal SLDC Processing

For each segment encoded, the segment weight code length and the rank length are added to the total number of bits encoded to give the compression ratio of either the entire image or, if requested, of each line.

3.2 Software Documentation

The software documentation for the SLDC encoder program is presented in this section and includes a structure chart, Nassi-Schneiderman flow charts for the major software modules, and descriptions of the functions and data storage methods associated with the encoder program. Also included in this section is the data flow diagram for the overall simulation project and Nassi-Schneiderman flow charts for the software modules that generate the Conditioned Image Files; this additional documentation is not directly related to the SLDC encoder program, but is presented here in order to convey a clearer picture of the software simulation of the SLDC compression algorithm.

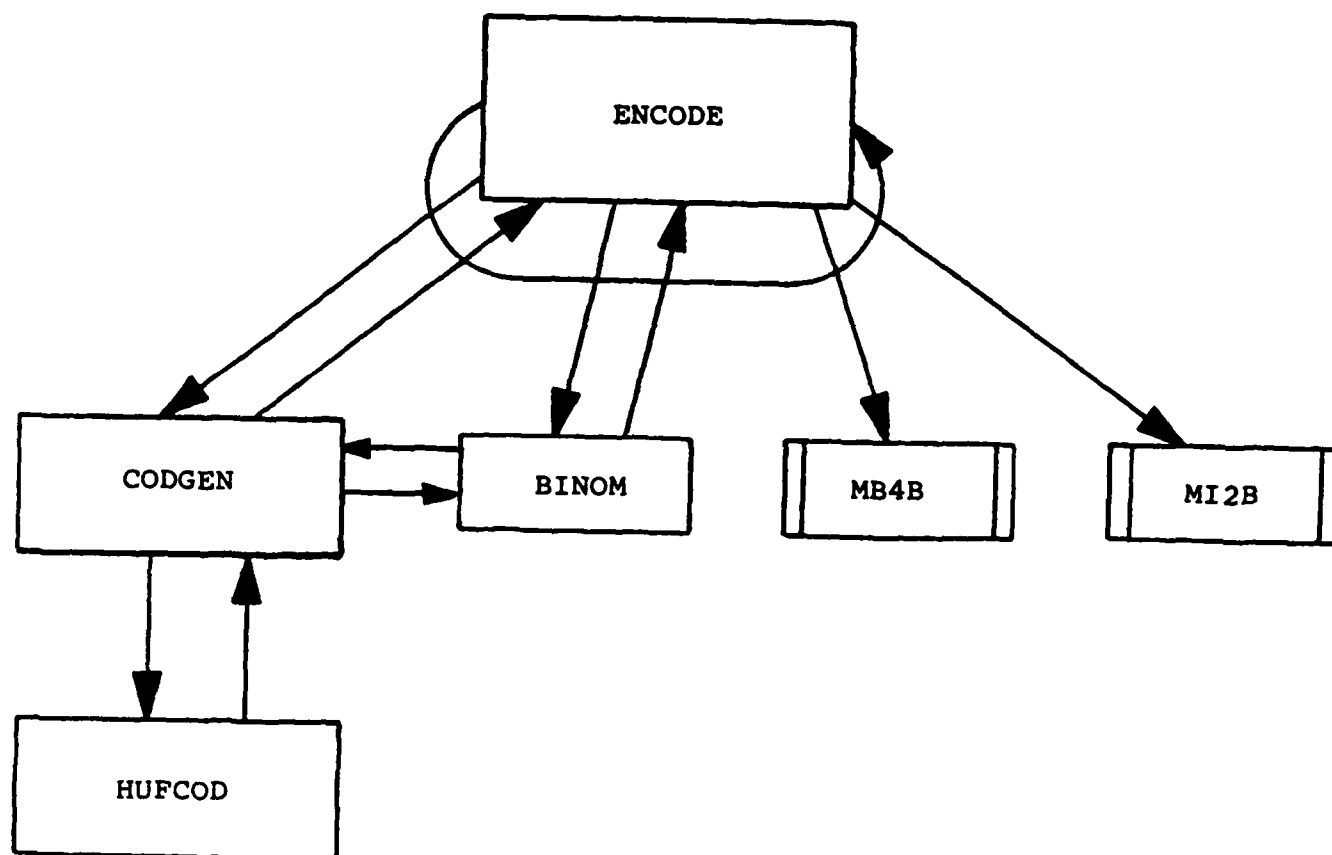


Figure 3.2 - Structure Chart for Module: ENCODE

Figure 3.3 - ENCODE.FTN - SLDC Encoding Program

OPEN files and read input parameters

DO for # records in file

READ Conditioned Scan Line (CSL) into buffer

Calculate line weight

Line weight = 0 ?

NO

YES

Put line weight in first 13 bits of ESL
Encoded Scan Line (ESL)

CALL CODGEN with lineweight and max weight
of segment

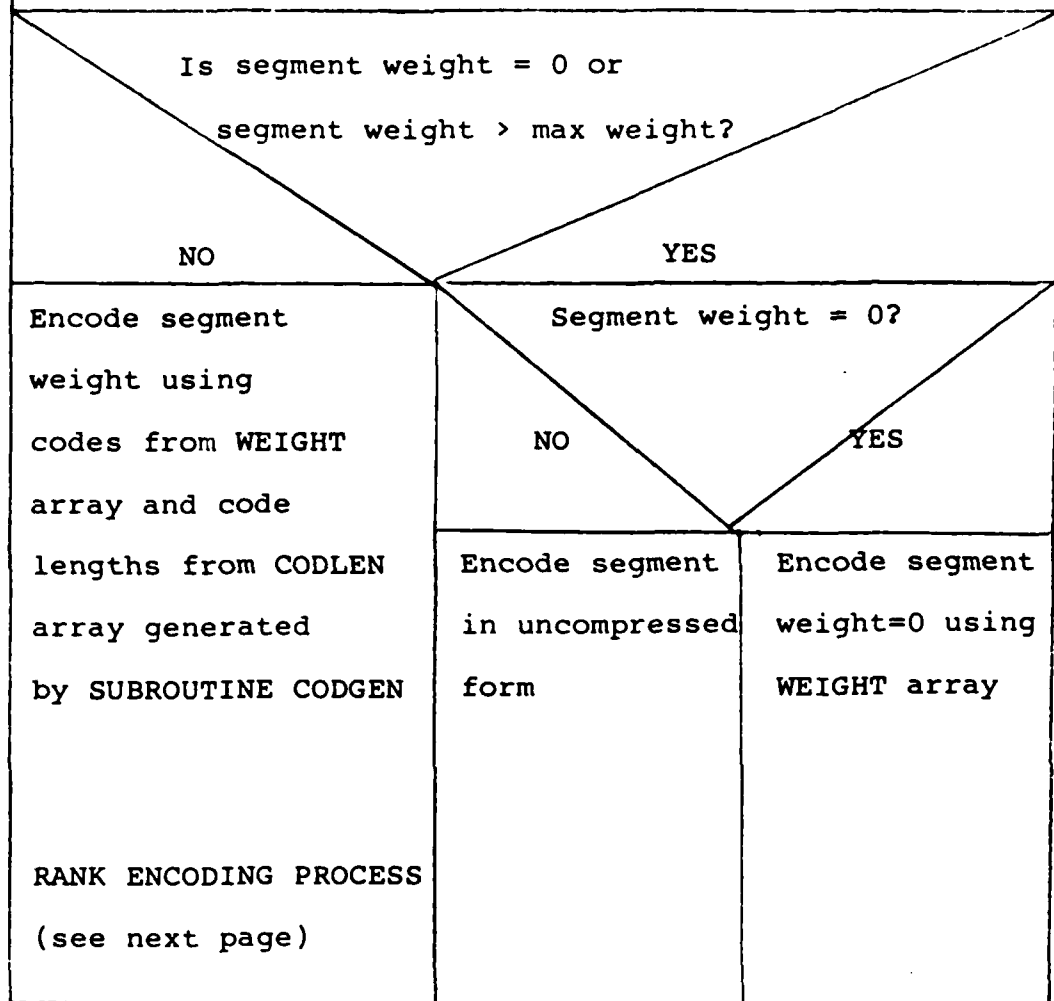
DO for # segments in CSL or
"1"s seen = line weight

Put segment of segment length from
line buffer into segment buffer

Calculate segment weight

Set first bit
of first word
of ESL and
increment
encoded line
weight count

Figure 3.3 - ENCODE.FTN - SLDC Encoding Program



END

Figure 3.3 - ENCODE.FTN - SLDC Encoding Program

RANK Encoding Process

DO for # bits in segment OR "1"s seen = segment weight

OR rest of segment is filled with "1"s

Put present buffer word into an integer

DO for # bits in segment OR "1"s seen = segment weight

OR rest of segment is filled with "1"s

Only "1"s remaining or "1"s seen
= segment weight?

NO

YES

Present bit = "1"?

YES

Set flag to exit loop

Add answer from RANK constant
equation described in SOW to
rank

Increment "1"s seen

Use RANK length equation described in SOW to determine RANK length

Encode RANK length number of bits to encode rank in ESL

Figure 3.4 - CODGEN.FTN - SLDC Huffman Coding Routine

DO for Weight = 0 to Max weight

Calculate probabilities for each weight
using EQ [A-4] in Appendix A

Assign Probability to Maxweight + 1
(1 - sum of probabilities)

Bubble sort probabilities in descending order keeping
track of original positions

CALL HUF COD to calculate codewords and codelengths
for probabilities

DO for weight = 0 to Maxweight +1

Put corresponding weight value back into proper weight
subscript of Weight array and Code length array
to provide easy access to codes and code lengths

RETURN

Figure 3.5 - HUFCOD.FTN - Huffman Codword Generator

DO for 0 to Max weight of segment

Add last two entries in column

Determine it's rank in next column

Place it in the next column at it's
proper rank

Mark the position in the position
array

Initialize codes and indicies

DO for Max weight down to 0

Add a bit to codeword at next column's
position according to position array

Set new bit of last entry in column to "0"

Set new bit of last entry in new column to "1"

RETURN

Program Documentation for module: ENCODE

PROGRAM: ENCODE

DESCRIPTION: This program encodes an input Conditioned Image File using SLDC encoding techniques. The program interactively inquires for, then accepts input parameters used in runs having different file sizes and design parameter sets. A summary of each run is printed including: names and sizes of files used and compression statistics. Options to create an output file (and/or) print line by line compression statistics are also included.

RUNSTRING: ENCODE, <INPUT NAME>, <OUTPUT NAME>

INPUT NAME Input Image File name

OUTPUT NAME Output Encoded File name.(Optional)

ORDER OF
INPUT PARAMETERS:

- 1) # Records to be output
- 2) Decision to create output file
- 3) Maximum weight for segment length
- 4) CCITT file number
- 5) File resolution
- 6) Arithmetic word length
- 7) # Words per input record
- 8) # Records per input file
- 9) Segment length for compression
- 10) Decision to print line by line compression statistics

MODULES CALLED:

CODGEN	Huffman code generation subroutine. Generates Huffman codes for each line in input file with line weight greater than zero.
BINOM	Binomial coefficient function
MI2B	Subroutine to place an integer into a given position in an array
MB4B	Subroutine to move parts of one array into parts of another array.

Program Documentation for module: ENCODE

MVBITS

BTEST,IBSET

FORTTRAN bit manipulating routines

NAMED COMMON
DESCRIPTIONS:

Block Name: ENCOD
Module Common to: CODGEN

Descriptions:

WEIGHT	Array of Huffman codes returned from CODGEN with array subscripts denoting segment weights
SEGL	Segment length design parameter used in EQ. [A-4].
NUMBER	Number of bits per CSL to be used in EQ. [A-4].
CODLEN	Code length array returned from CODGEN with values corresponding to those of WEIGHT array

Subroutine Documentation for module: CODGEN

SUBROUTINE: CODGEN

MODULES
CALLED FROM: ENCODE and DECODE

PURPOSE: This subroutine generates probabilities for each possible weight from 0 upto and including the maximum weight allowable for a given segment+1 passed by parameter. These probabilities are then used by SUBROUTINE HUF COD to generate codes and code lengths for these probabilities.

MODULES CALLED:

BINOM	Binomial coefficient function
HUF COD	Subroutine which assigns variable length Huffman codes and code lengths to segment weights which are arranged in most probable to least probable order.

CALLING FORMAT: CALL CODGEN(LNWGT,MAXWGT)

ARGUMENT DESCRIPTIONS:

LNWGT	Current CSL weight generated by ENCODE. LNWGT is used by EQ. [A-4].
MAXWGT	Maximum segment weight encodable for design parameter segment length taken from TABLE [A-1] MAXWGT is also used by EQ. [A-4].

NAMED COMMON DESCRIPTIONS:

Block Name: ENCOD
Module Common to: ENCODE

Descriptions:

WEIGHT	Array of Huffman codes returning to ENCODE with array subscripts denoting segment weights
--------	---

Subroutine Documentation for module: CODGEN

SEGL	Segment length design parameter used in EQ. [A-4].
NUMBPL	Number of bits per CSL to be used in EQ. [A-4].
CODLEN	Code length array returning to ENCODE with values corresponding to those of WEIGHT array

Block Name:	HUFBLK
Module Common to:	HUFCOD

Descriptions:

PROB	Probability array containing probabilities of segment weights of a given CSL.
CODE	Array returned holding huffman codes in most probable to least probable order
MAXWT	Maximum weight of a segment. (described above)
CDWLEN	Code length array matching values in CODE array.

Subroutine Documentation for module: HUF COD.FTN

SUBROUTINE: HUF COD

MODULE
CALLED FROM: CODGEN

PURPOSE: This subroutine takes probabilities generated by CODGEN and determines the positions for a right to left expansion of the huffman coding table according to the values of the probabilities.

CALLING FORMAT: CALL HUF COD

NAMED COMMON
DESCRIPTIONS:

Block Name: HUFBLK
Module Common to: CODGEN

Descriptions:

PROB	Probability array containing probabilities of segment weights of a given CSL.
CODE	Array returning huffman codes in most probable to least probable order
WGTNUM	Maximum weight of a segment. (described above)
CDWLEN	Code length array matching values in CODE array.

Subroutine Documentation for module: BINOM.FTN

FUNCTION: BINOM

MODULES
CALLED FROM: CODGEN, ENCODE, DECODE

PURPOSE: This function uses an algorithm found in CACM to calculate the binomial coefficient given two parameters. The maximum value returned is 2^{31} which is the maximum value representable in a FORTRAN DOUBLE INTEGER.

CALLING FORMAT: $n = \text{BINOM}(N, M)$ where n is a single or double integer

ARGUMENT
DESCRIPTIONS:

N, M N objects taken M at a time

XMB4BE.LIB::DAF4LIB
FORTRAN 77 Subroutines
Author: S. J. Urban
8/5/85

DIS FORTRAN 77 bit handling routines

FUNCTION:

Provides facilities for storing integers, in packed form, in variables and arrays, and for retrieving them at a later time.

CALLS:

CALL MI2B(I2, BA, JB, NB)

Stores the INTEGER*4 I2 into BA, starting at the JBth bit, and occupying NB bits.

I4B(BA, JB, NB)

Retrieves (as its functional value) the INTEGER*4 integer stored in BA, starting at the JBth bit, and occupying NB bits. I4B returns a INTEGER*4 integer value and must be declared as such in the calling routine.

CALL MB4B(TBA, JTB, NB, FBA, JFB)

Replaces JTBth through the (JTBth + NB - 1)st bits of TBA with the JFBth through the (JFB + NB - 1)st bits of FBA.

ARGUMENTS:

I2	- INTEGER*4 variable or array element
JTB,JFB,JB	- INTEGER*2 starting bit position for string; must be > 0.
NB	- INTEGER*2 no. of bits in a string (i.e. string size must be > 0.
TBA,FBA,BA	- INTEGER*2 or INTEGER*4 arrays used for storing "packed" bit strings.

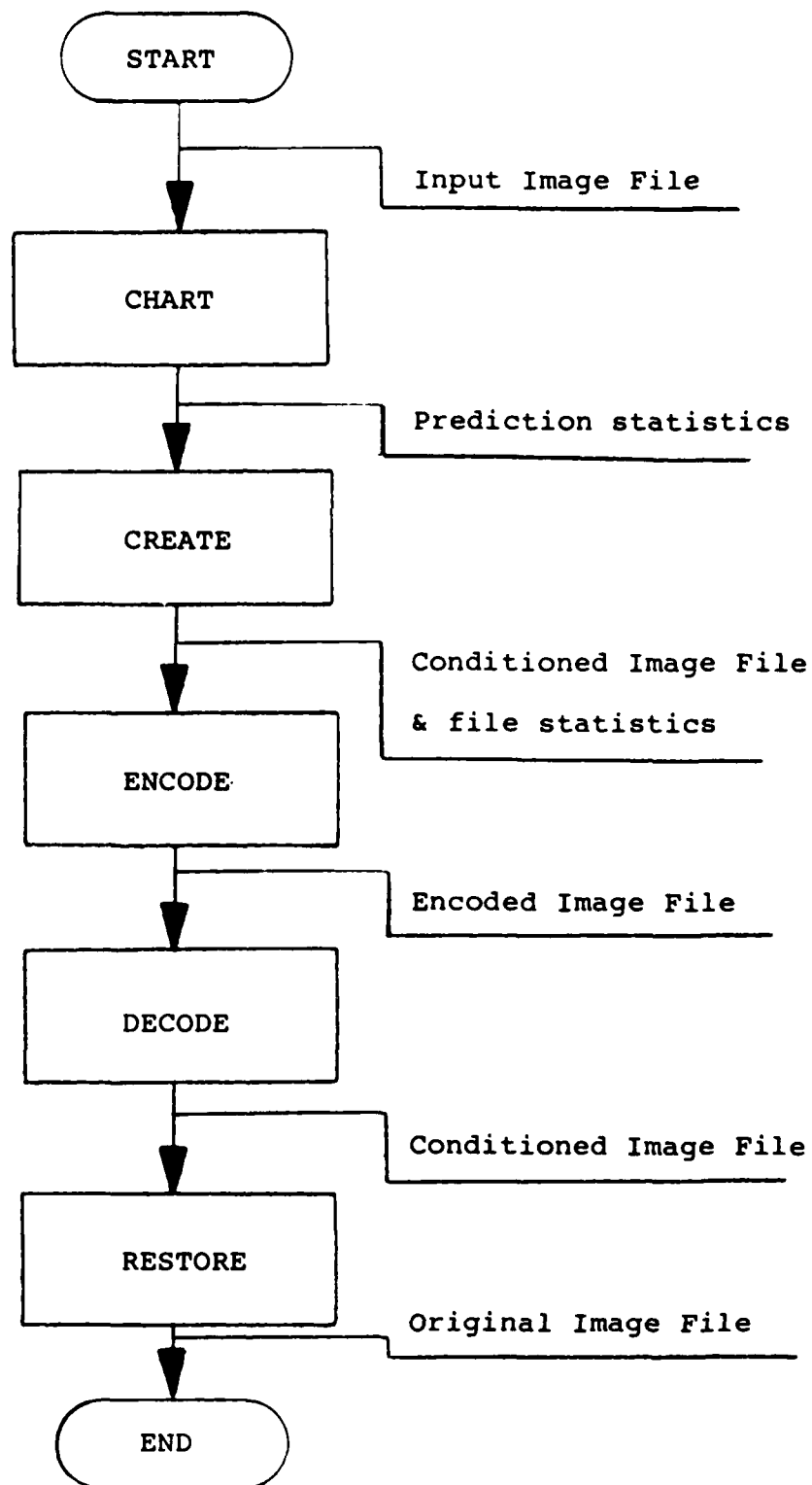


Figure 3.6 - SLDC Data Flow Diagram

Figure 3.7 - CHART.FTN - SLDC Probability Statistics Generator

OPEN files and read input parameters

Initialize all count arrays and buffers

DO for # records in file

DO for # words in input record

DO for # bits per word

Test for value of current bit being
examined

Test for neighboring bit values

Search template array for current
neighbor template

Increment count array for total,
and black white count of current
neighbor template

Print all statistics

Figure 3.7 - CHART.FTN - SLDC Probability Statistics Generator

WRITE proper prediction (1 or 0) for each neighbor template
to probability file for use in later programs

E N D

Figure 3.8 - CREATE.FTN - SLDC CIF Generating Program

OPEN files and read input parameters

READ Probability Data Generated from PROGRAM CHART

Initialize buffers and count arrays

DO for # records in file

DO for # words in input record

DO for # bits per word

Test for neighboring bit values

Search probability data for proper
neighbor template and respective
prediction

Is prediction correct?

NO

Set current bit to "1"

Increment lineweight count, 32 bit segment weight
count, and 64 bit segment weight count

Figure 3.8 - CREATE.FTN - SLDC CIF Generating Program

Reach end of 32 bit segment?

YES

Increment proper 32 bit weight subscript in 32 bit
count array

Reach end of 64 bit segment?

YES

Increment proper 64 bit weight subscript in 64 bit
count array

Put present output word in output buffer

Increment proper line weight subscript in line weight
count array

Add line weight to file weight

WRITE output buffer to output file

Print out all statistics

CLOSE files

E N D

Figure 3.9 - RESTORE.FTN - SLDC Original Restoration Program

OPEN files and read input parameters

READ Probability Data Generated from PROGRAM CHART

Initialize buffers and count arrays

DO for # records in file

DO for # words in input record

DO for # bits per word

Test current bit being examined

Test for neighboring bit values

Search probability data for proper
neighbor template and respective
prediction

Is current bit "1"?

YES

NO

Place bit opposite of
prediction bit in
current bit position

Place prediction bit in
current bit position

Figure 3.9 - RESTORE.FTN - SLDC Original Restoration Program

Put present output word in output buffer

WRITE output buffer to output file

CLOSE files

E N D

4.0 SLDC Decoder Software Design

The SLDC decoder program reconstructs the Conditioned Image Files (CIF's) compressed in the previous subtask by employing the identical codeword generator and rank encoding procedure employed in the SLDC encoder program. A brief description of the decoding process is presented here, along with the documentation for the software modules associated with the decoder program. The operating instructions for the decoder program are presented in Appendix B; the code listings appear in Appendix C.

4.1 Functional Description

Decoding is also done one line at a time. Each ESL is read into a buffer and the first bit in the line (bit 15 in word 1) is checked. If the bit is set to one (1), the line weight is zero and all bits in the decoded CSL are set to zero (0). If the bit is not set to one (1), the first 13 bits of the ESL are interpreted as the line weight and the Huffman coding subroutine is called to generate Huffman codes for all possible segment weights for that particular line weight. The program proceeds by decoding one segment at a time; a number of bits equal to the maximum segment weight codeword are read from the ESL into a buffer. The segment weight is then decoded by comparing each code in the Huffman code array with the same number of bits in the codeword buffer until the code is found.

If the segment weight is found to be zero, the next n bits in the decoded CSL, where n is the segment length, are set to zero (0). If the segment weight is found to be greater than the maximum allowable

weight, the segment has been "transmitted" uncompressed, and the next n bits in the ESL are placed directly into the decoded CSL, where, again, n is the segment length.

For the remaining condition, in which the segment weight is greater than zero and less than the maximum allowable, the length of the rank can be calculated by using the segment weight in the rank length equation. The rank is then determined from the integer value of the next r bits in the ESL, where r is the determined rank length. The segment is reconstructed by placing a one (1) in the output segment buffer whenever the difference between the rank and the calculated binomial coefficient (see Appendix A) is positive; when a one (1) is found, the rank is decremented by an amount equal to the coefficient. This rank decoding process continues until the entire rank is examined, or the rank is zero after the decrement takes place. The output segment buffer is then placed in the decoded CSL. The segment decoding continues until all segments in the line have been decoded, or until the sum of the segment weights of the segments placed in the decoded CSL is equal to the line weight.

4.2 Software Documentation

The software documentation for the SLDC decoder program is presented in this section and includes a structure chart, a Nassi-Schneiderman flow chart, and a description of the functions and data storage methods associated with the main software module, DECODE. All other modules associated with the decoder program are identical to those of the encoder program; the documentation for these modules appears in section 3.2.

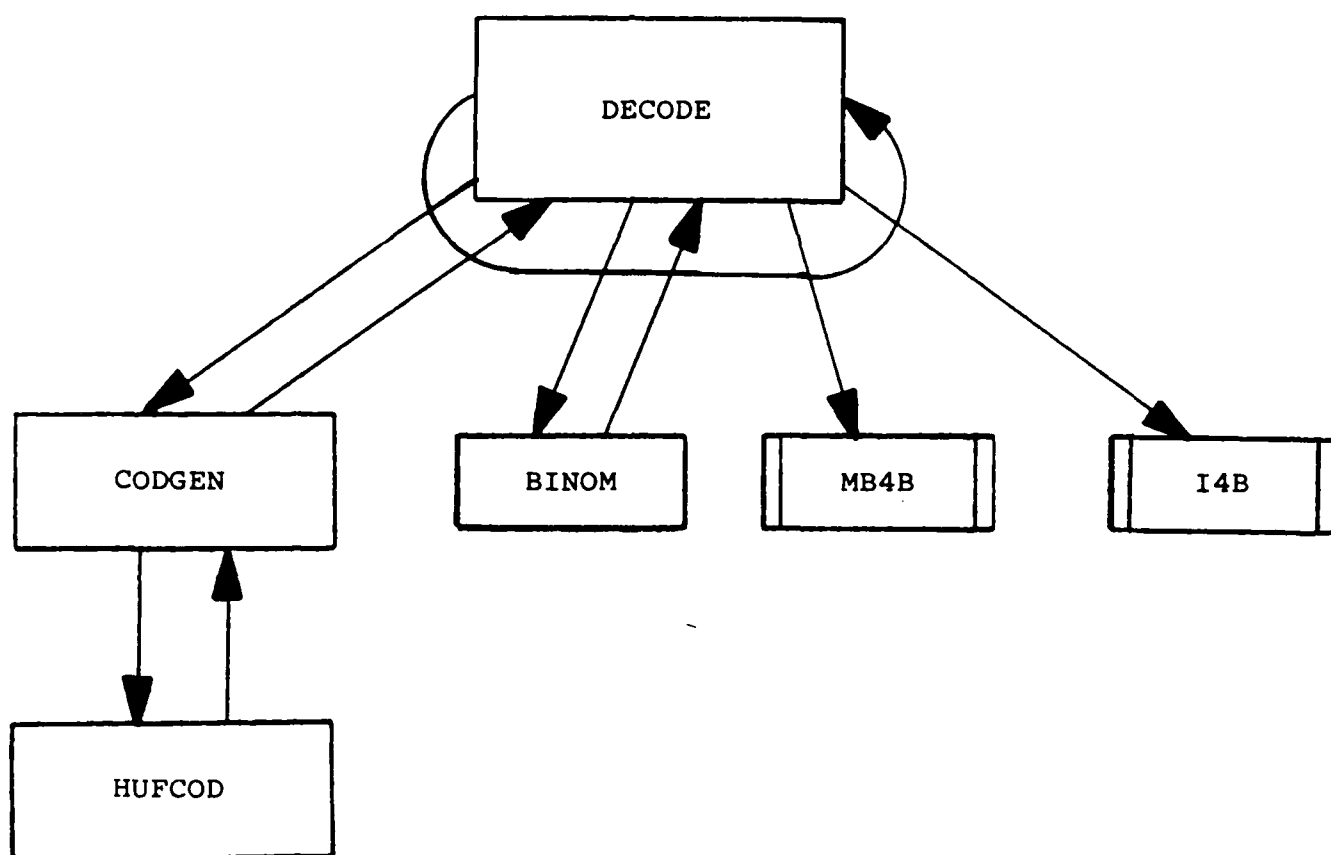


Figure 4.1 - Structure Chart for Module: DECODE

Figure 4.1 - DECODE.FTN - SLDC Decoding Program

OPEN files and read input parameters

DO for # records in file

READ ESL into buffer

First bit in first word of ESL set ?

NO

CALL CODGEN with lineweight and max weight of segment

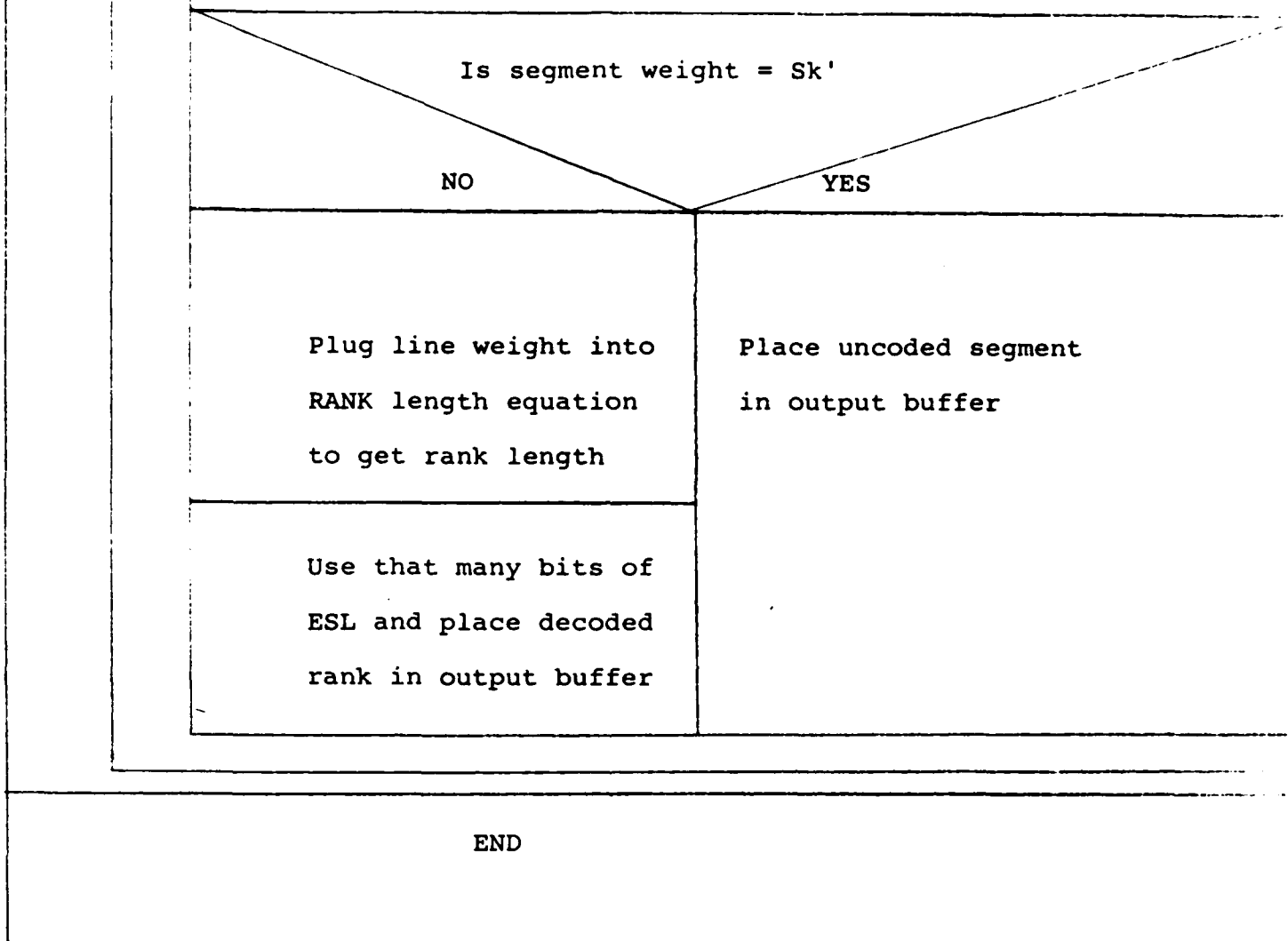
DO WHILE # of "1"s put in output line < line weight

READ in code buffer (double integer from ESL)

DO WHILE segment weight code is not found

Check current code in array with same
number of bits in code buffer

Figure 4.1 - DECODE.FTN - SLDC Decoding Program



Program Documentation for module: DECODE

PROGRAM: DECODE

DESCRIPTION: This program decodes an input Encodes Image File coded using SLDC encoding techniques. The program interactively inquires for, then accepts input parameters used in runs having different file sizes and design parameter sets. A summary of each run is printed including names and sizes of files used and design parameters.

RUNSTRING: DECODE,<INPUT NAME>,<OUTPUT NAME>

INPUT NAME Input Image File name

OUTPUT NAME Output Conditioned Image File

ORDER OF
INPUT PARAMETERS:

- 1) # Words per output record
- 2) # Records to be output
- 3) Arithmetic word length
- 4) # Words per input record
- 5) # Records per input file
- 6) Segment length for compression
- 7) Maximum weight per segment

MODULES CALLED:

CODGEN Huffman code generation subroutine. Generates Huffman codes for each line in input file with line weight greater than zero.

BINOM Binomial coefficient function

I4B Function to extract an integer from a given position in an array

MB4B Subroutine to move parts of one array into parts of another array.

MVBITS,IBITS
BTEST,IBSET FORTRAN bit manipulating routines

NAMED COMMON
DESCRIPTIONS:

Block Name: ENCOD

Program Documentation for module: DECODE

Module Common to: CODGEN

Descriptions:

WEIGHT	Array of Huffman codes returned from CODGEN with array subscripts denoting segment weights
SEGL	Segment length design parameter used in EQ. [A-4].
NUMBER	Number of bits per CSL to be used in EQ. [A-4].
CODLEN	Code length array returned from CODGEN with values corresponding to those of WEIGHT array

5.0 SLDC Compression Results and Analysis

Simulation runs made on each image using the SLDC technique included design parameters of both 16 and 32 bit arithmetic word lengths using segment lengths from 8 to 64, in increments of 8. The parameters were chosen to experiment with many combinations of word lengths and segment lengths to determine the most efficient parameter set for each image at each resolution.

The results from the SLDC compression simulation runs showed the segment lengths using the 32 bit arithmetic word to have consistently higher compression on each image tested at all four resolutions, except for segment lengths of 8 and 16, where compression is equal due to the design of the algorithm. The higher compression in the 32 bit word simulation runs is related to the maximum weight constraint difference between the 16 and 32 bit word lengths, where compression of the segment to be encoded is bypassed if the weight of the segment exceeds the maximum weight constraint. The compression bypass involves skipping the rank encoding part of the algorithm when the segment weight is higher than the maximum weight allowable for that segment. Another direct result of the compression bypass is a difference in processing time, which is consistently lower in the 16 bit word simulation runs due to the processing skipped when the rank encoding is omitted. The processing times of all of the simulation runs appear to be directly related to the maximum weight constraint and almost independent of segment length, which illustrates which parameter determines the difference in the complexity of the algorithm between simulation runs.

The comparison of the overall compression between images was as expected, with the English letter compressing the best, followed by the French journal, and then the Kanji text; the compression statistics are presented in Tables 5.1 through 5.12, and are illustrated graphically in Figures 5.1 through 5.12 (In all of the compression graphs, the □ indicates results of the 16 bit word length runs and the + indicates the results of the 32 bit word length runs.). The French and English document compression curves are quite similar, with maximum compression of both 16 and 32 bit word length runs occurring at the same segment lengths. It appears that the similarities between the two compression curves are a result of the similar probability distributions of the two files along with similar CIF constructions. The Kanji compression curve is different than those of the French and English documents. The Kanji 32 bit word curve is without the "peaking" effect demonstrated by the French and English curves. The maximum compression point is virtually indistinguishable in the 24 to 56 bit segment range, as compared to the definite peaks in both the English and French curves.

The maximum compression point varies with resolution. As resolution increases, the segment length of the maximum point of the 16 and 32 bit word length run curves increases. This can be described as a "right shift" of the peaks in the graphs. The shift is due to more segments with lower segment weights as the resolution increases; therefore, fewer segments exceed the maximum weights at higher segment lengths, and this leads to optimum compression at higher segment lengths. The suggested set of parameters to achieve maximum compression, taking processing time into consideration, is presented in Table 5.13.

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	15.31	10.31
16	16	18.49	11.22
16	24	19.41	9.41
16	32	17.94	9.23
16	40	15.67	9.04
16	48	14.77	8.56
16	56	14.14	8.50
16	64	13.33	8.23
32	24	19.53	14.04
32	32	20.03	17.21
32	40	20.17	11.34
32	48	19.89	10.22
32	56	19.11	10.03
32	64	17.61	9.39

Table 5.1 - Compression Stats, English Letter - 200 LPI

SEGMENT LENGTH VS. COMPRESSION

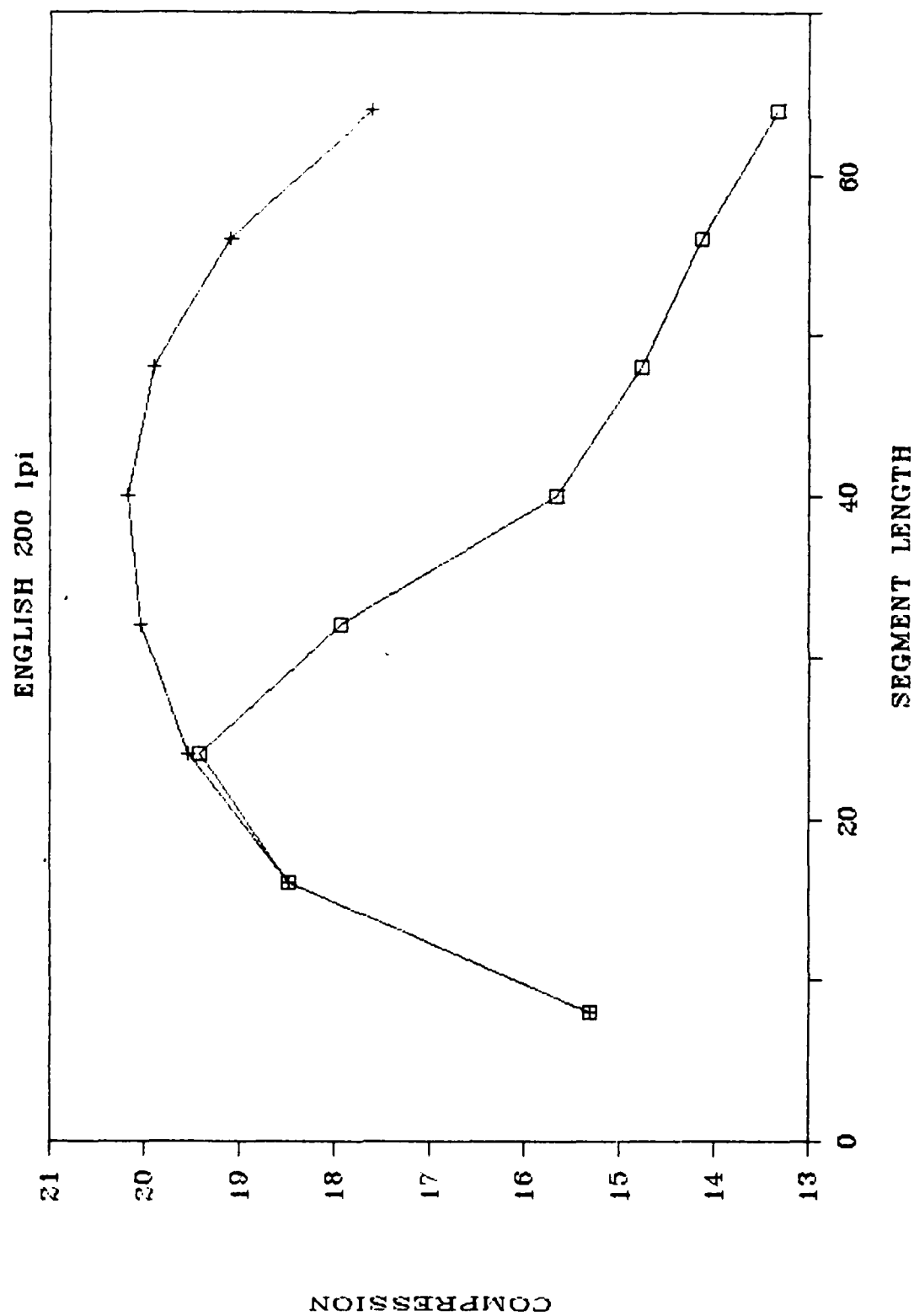


Figure 5.1

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	8.38	17.22
16	16	10.35	18.14
16	24	10.89	15.29
16	32	10.34	14.48
16	40	9.09	14.01
16	48	8.59	13.43
16	56	8.21	13.10
16	64	7.88	12.38
32	24	11.12	24.19
32	32	11.48	30.45
32	40	11.61	21.22
32	48	11.44	20.30
32	56	11.03	20.00
32	64	10.27	19.12

Table 5.2 - Compression Stats, French Journal - 200 LPI

SEGMENT LENGTH VS. COMPRESSION

FRENCH 200 lpi

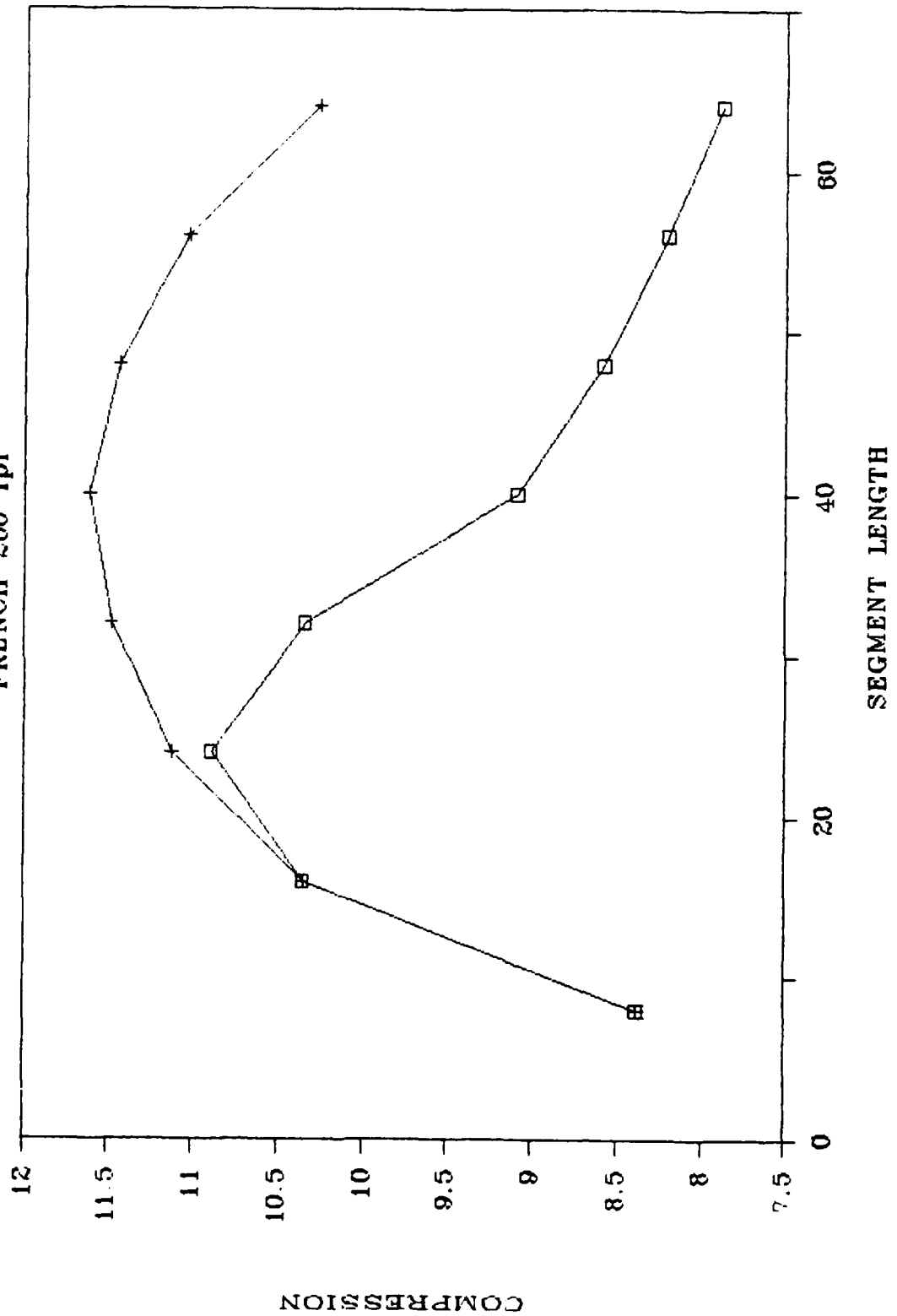


Figure 5.2

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	4.88	22.19
16	16	5.41	24.32
16	24	5.33	19.31
16	32	5.00	19.00
16	40	4.06	18.39
16	48	3.68	17.56
16	56	3.40	17.22
16	64	3.15	16.49
32	24	5.41	31.22
32	32	5.44	37.51
32	40	5.44	22.17
32	48	5.41	20.41
32	56	5.37	19.12
32	64	4.99	18.41

Table 5.3 - Compression Stats, Kanji Text - 200 LPI

SEGMENT LENGTH VS. COMPRESSION

KANJI 200 lpi

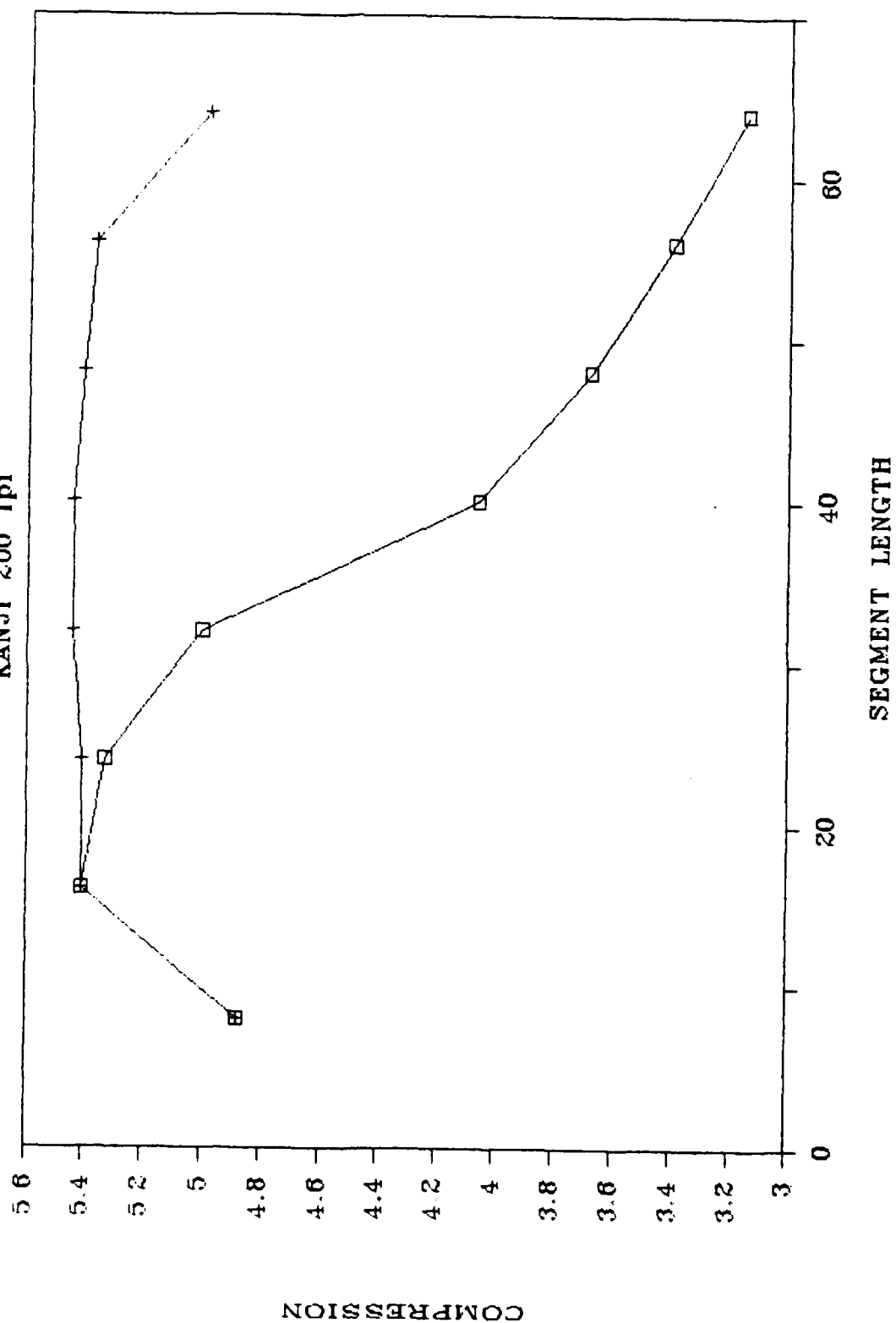


Figure 5.3

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	17.08	13.04
16	16	20.88	14.06
16	24	21.97	10.56
16	32	20.69	9.56
16	40	18.07	9.46
16	48	16.84	9.14
16	56	15.93	9.21
16	64	15.19	9.06
32	24	22.30	17.02
32	32	22.87	20.10
32	40	23.16	12.25
32	48	22.93	11.57
32	56	22.33	11.28
32	64	20.55	10.44

Table 5.4 - Compression Stats, English Letter - 240 LPI

SEGMENT LENGTH VS. COMPRESSION

ENGLISH 240 LPI

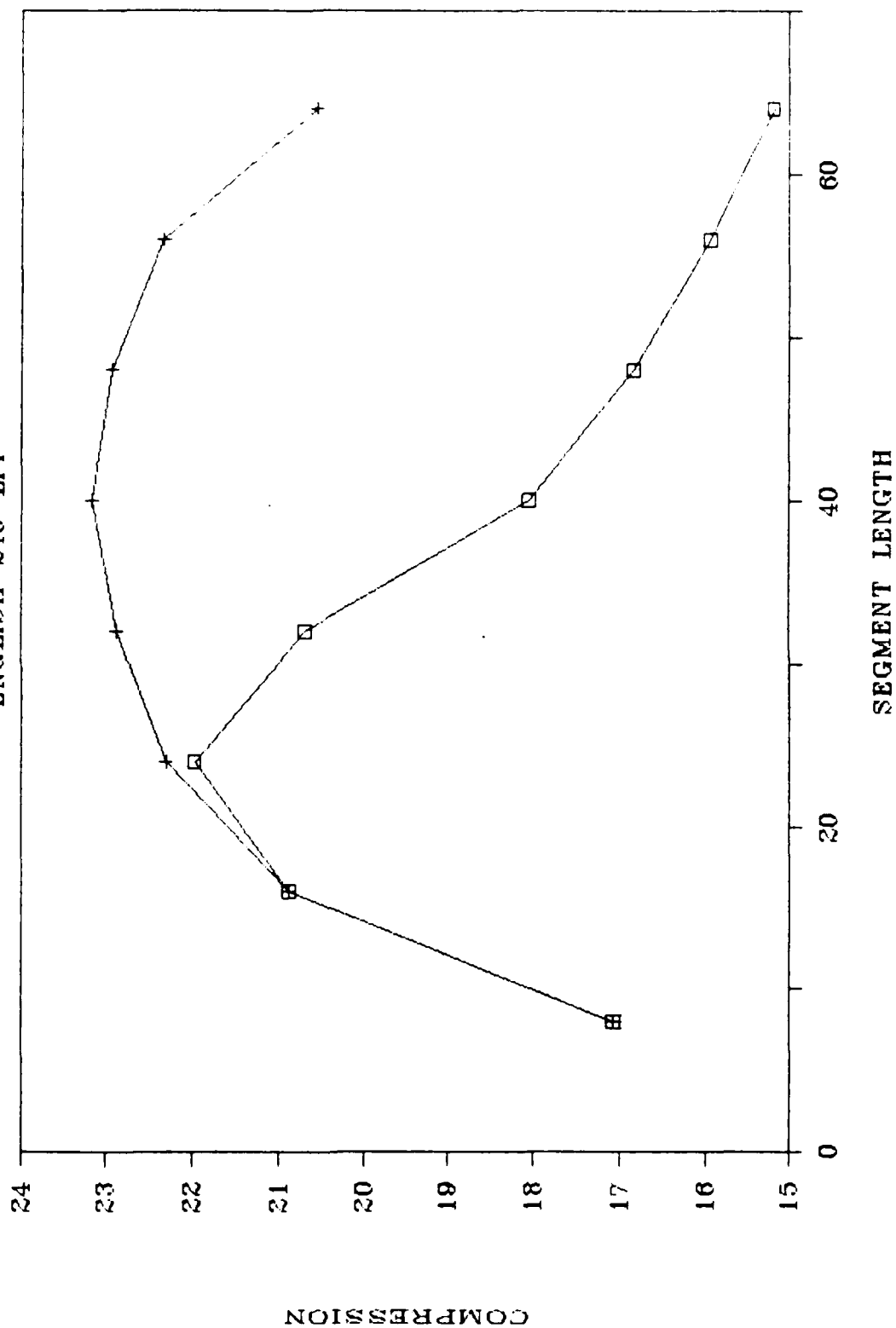


Figure 5.4

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	8.77	21.25
16	16	11.25	22.37
16	24	12.01	16.49
16	32	11.52	15.08
16	40	10.16	14.18
16	48	9.62	13.43
16	56	9.16	13.51
16	64	8.70	13.21
32	24	12.21	27.59
32	32	12.69	33.42
32	40	12.90	19.11
32	48	12.83	18.11
32	56	12.52	17.37
32	64	11.65	16.45

Table 5.5 - Compression Stats, French Journal - 240 LPI

SEGMENT LENGTH VS. COMPRESSION

FRENCH 240 LPI

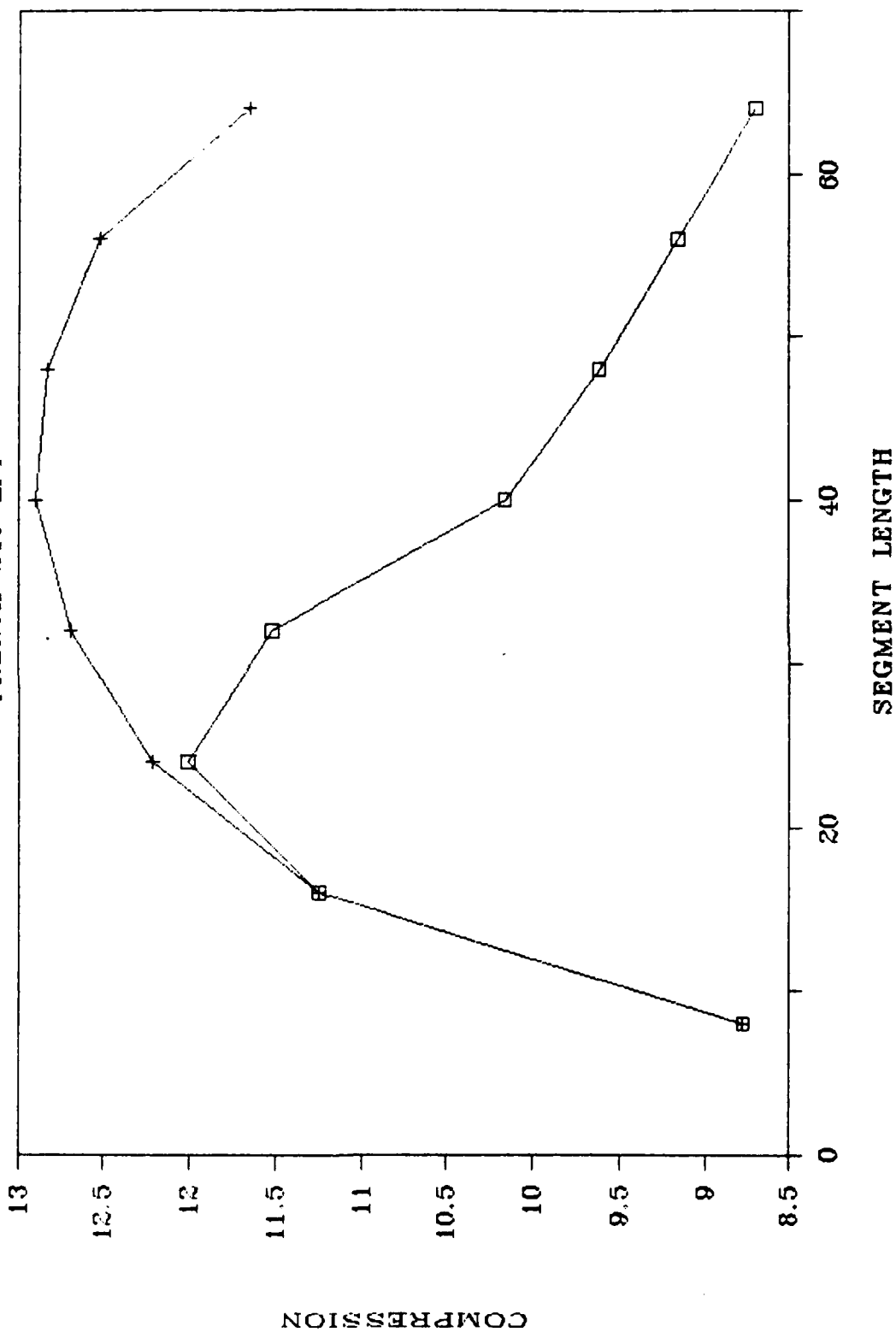


Figure 5.5

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	5.33	28.22
16	16	6.18	29.20
16	24	6.11	22.14
16	32	5.75	20.07
16	40	4.78	19.07
16	48	4.30	18.04
16	56	3.96	18.04
16	64	3.63	17.23
32	24	6.18	36.00
32	32	6.20	43.20
32	40	6.23	25.37
32	48	6.20	24.31
32	56	6.22	24.10
32	64	5.84	22.17

Table 5.6 - Compression Stats, Kanji Text - 240 LPI

SEGMENT LENGTH VS. COMPRESSION

KANJI 240 LPI

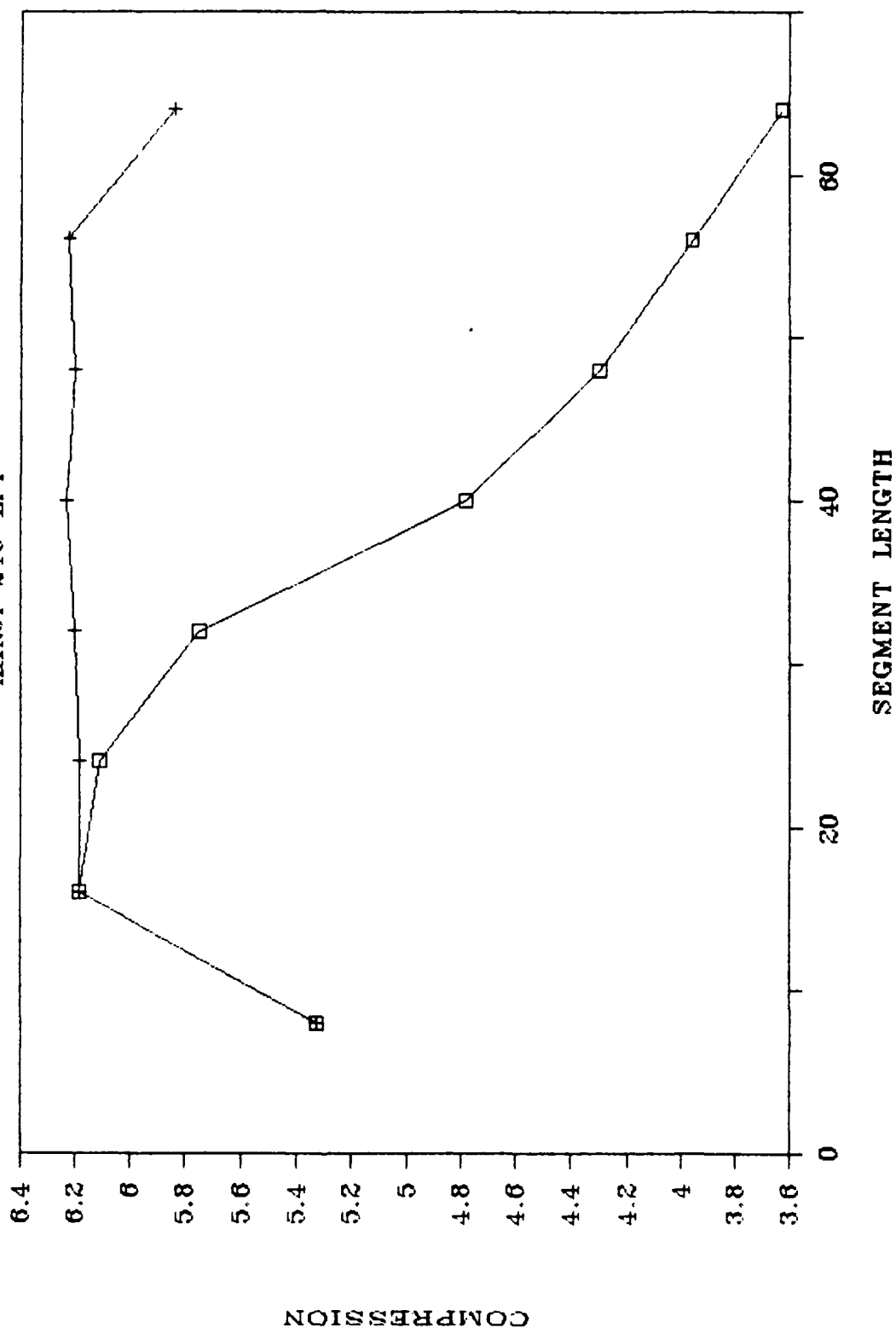


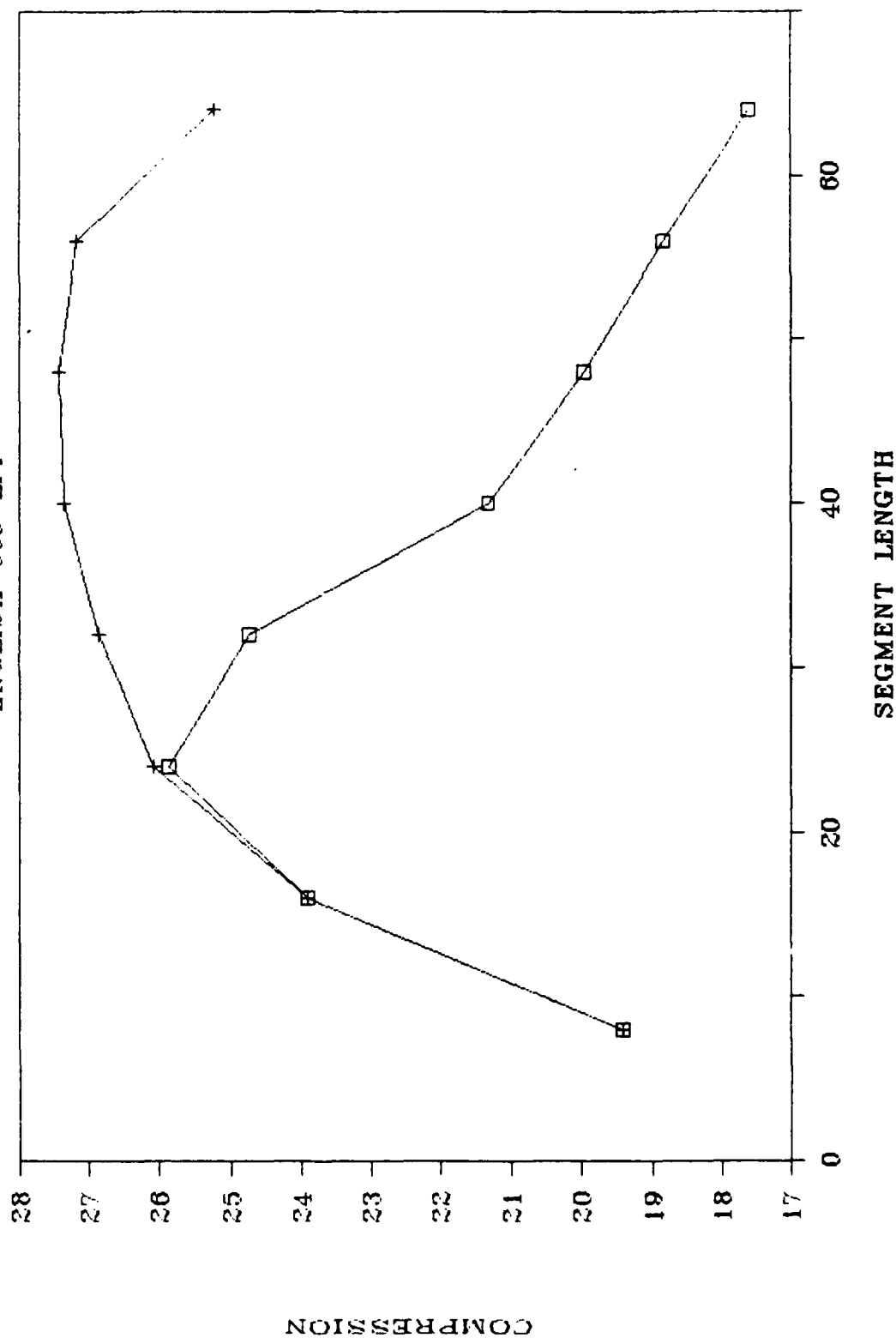
Figure 5.6

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	19.43	18.82
16	16	23.91	20.14
16	24	25.86	16.12
16	32	24.73	15.03
16	40	21.32	14.32
16	48	19.97	14.08
16	56	18.83	14.18
16	64	17.60	13.58
32	24	26.08	23.28
32	32	26.86	27.20
32	40	27.35	18.07
32	48	27.43	17.12
32	56	27.17	16.53
32	64	25.24	16.03

Table 5.7 - Compression Stats, English Letter - 300 LPI

SEGMENT LENGTH VS. COMPRESSION

ENGLISH 300 LPI



COMPRESSION

Figure 5.7

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	9.58	35.42
16	16	12.78	34.57
16	24	13.91	32.49
16	32	13.51	27.21
16	40	11.89	26.37
16	48	11.21	25.46
16	56	10.64	25.31
16	64	10.10	24.45
32	24	14.04	41.18
32	32	14.67	45.25
32	40	15.02	38.37
32	48	15.10	37.41
32	56	14.93	36.27
32	64	13.94	35.34

Table 5.8 - Compression Stats, French Journal - 300 LPI

SEGMENT LENGTH VS. COMPRESSION

FRENCH 300 LPI

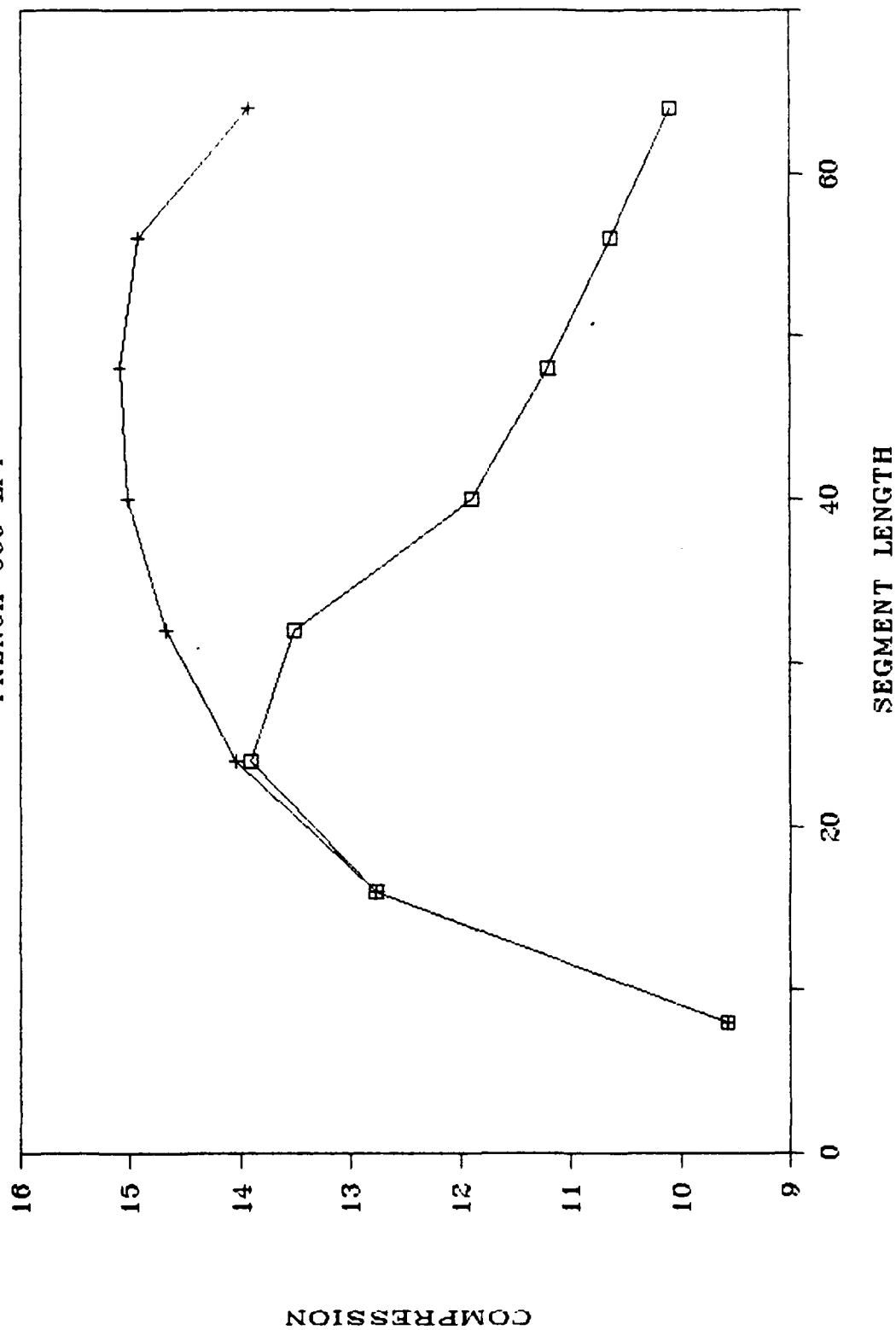
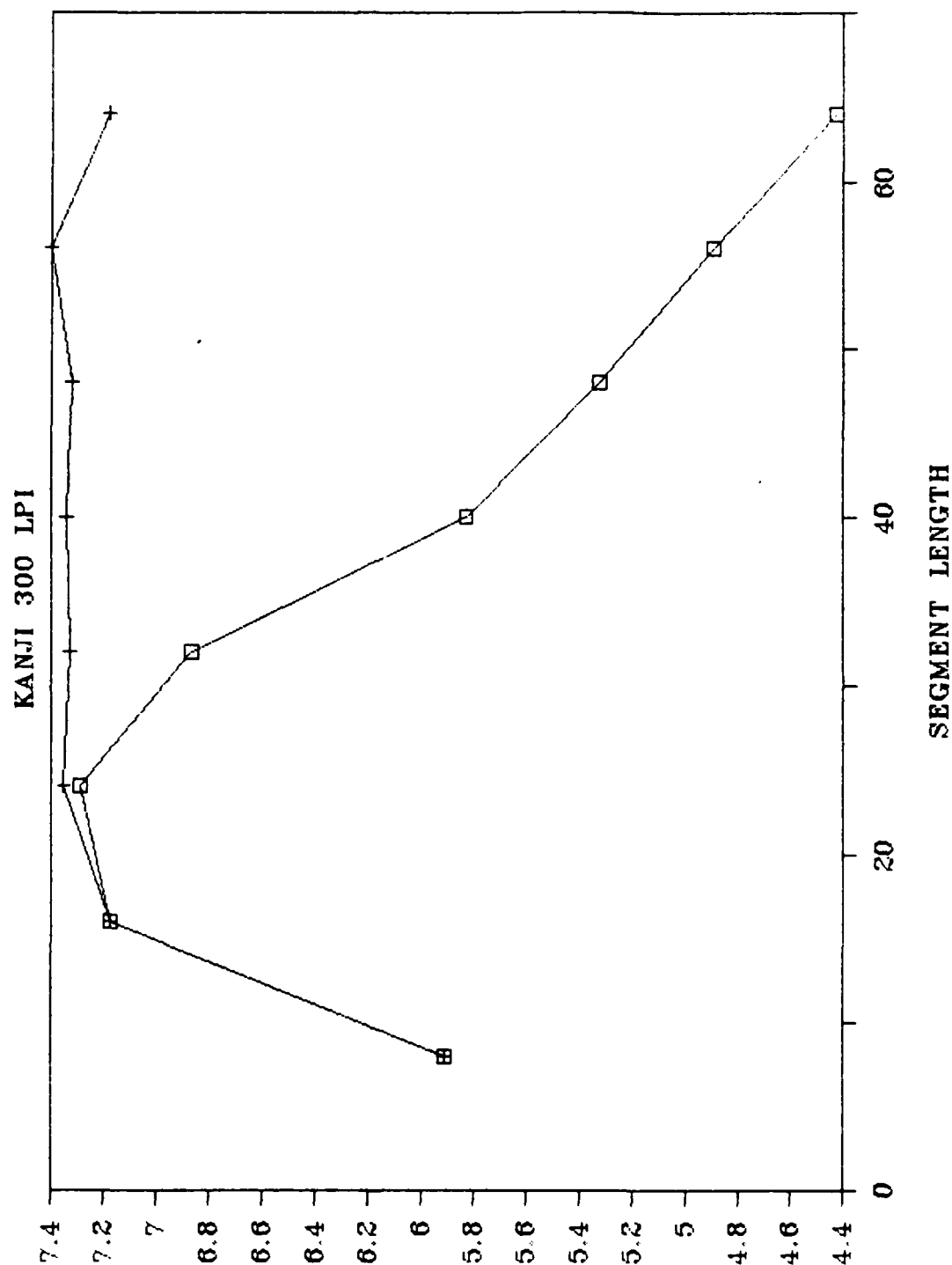


Figure 5.8

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	5.91	40.57
16	16	7.17	41.09
16	24	7.29	32.21
16	32	6.86	29.40
16	40	5.83	29.34
16	48	5.33	27.55
16	56	4.90	28.01
16	64	4.43	26.22
32	24	7.35	49.36
32	32	7.33	57.34
32	40	7.34	36.03
32	48	7.32	34.07
32	56	7.40	34.04
32	64	7.18	32.41

Table 5.9 - Compression Stats, Kanji Text - 300 LPI

SEGMENT LENGTH VS. COMPRESSION



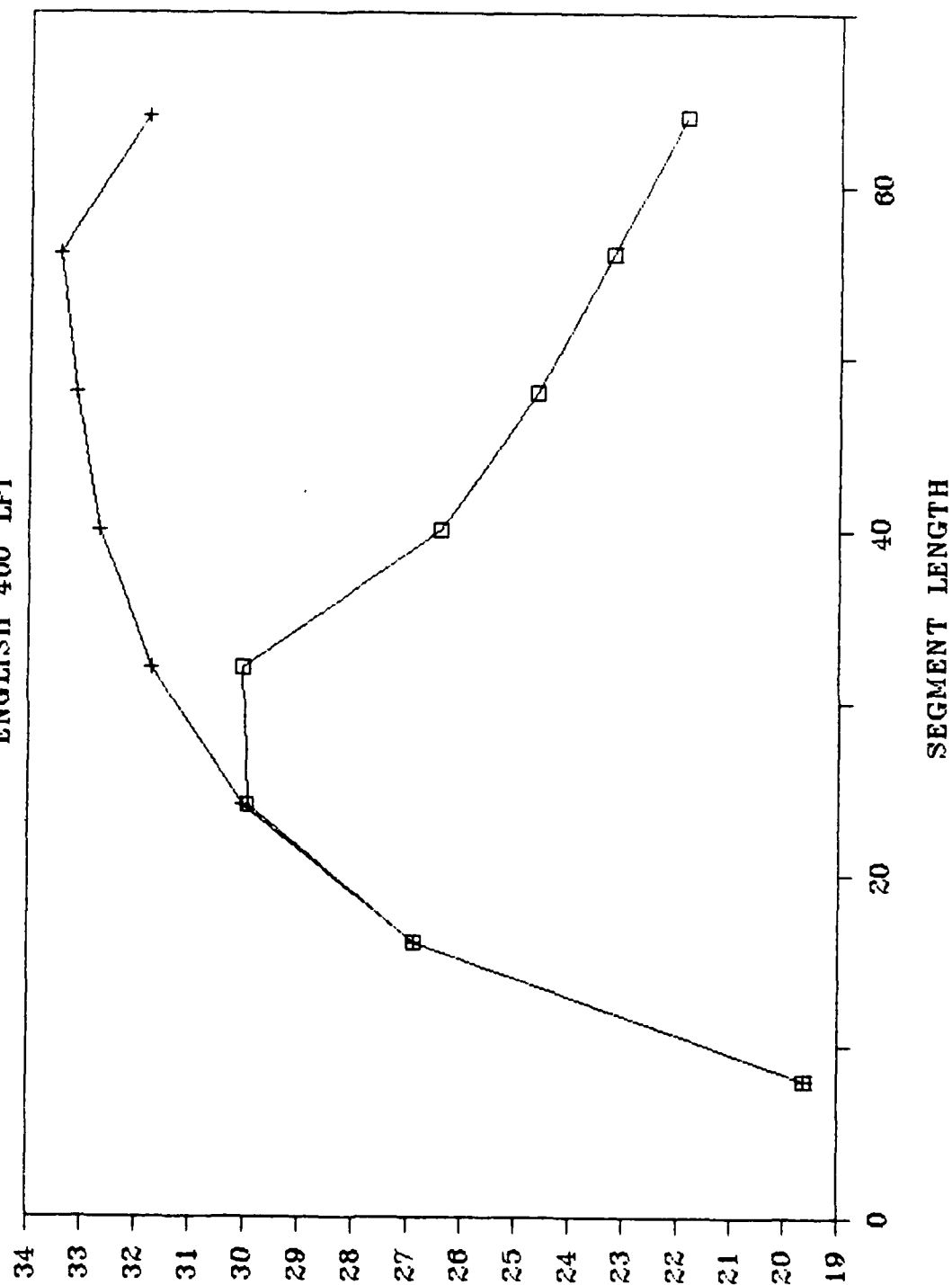
COMPRESSION
Figure 5.9

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	19.63	35.39
16	16	26.88	35.05
16	24	29.95	29.11
16	32	30.04	27.24
16	40	26.40	26.37
16	48	24.63	25.40
16	56	23.24	26.00
16	64	21.88	25.23
32	24	30.04	39.48
32	32	31.71	45.02
32	40	32.66	31.15
32	48	33.11	30.01
32	56	32.45	29.52
32	64	31.81	28.31

Table 5.10 - Compression Stats, English Letter - 400 LPI

SEGMENT LENGTH VS. COMPRESSION

ENGLISH 400 LPI



COMPRESSION

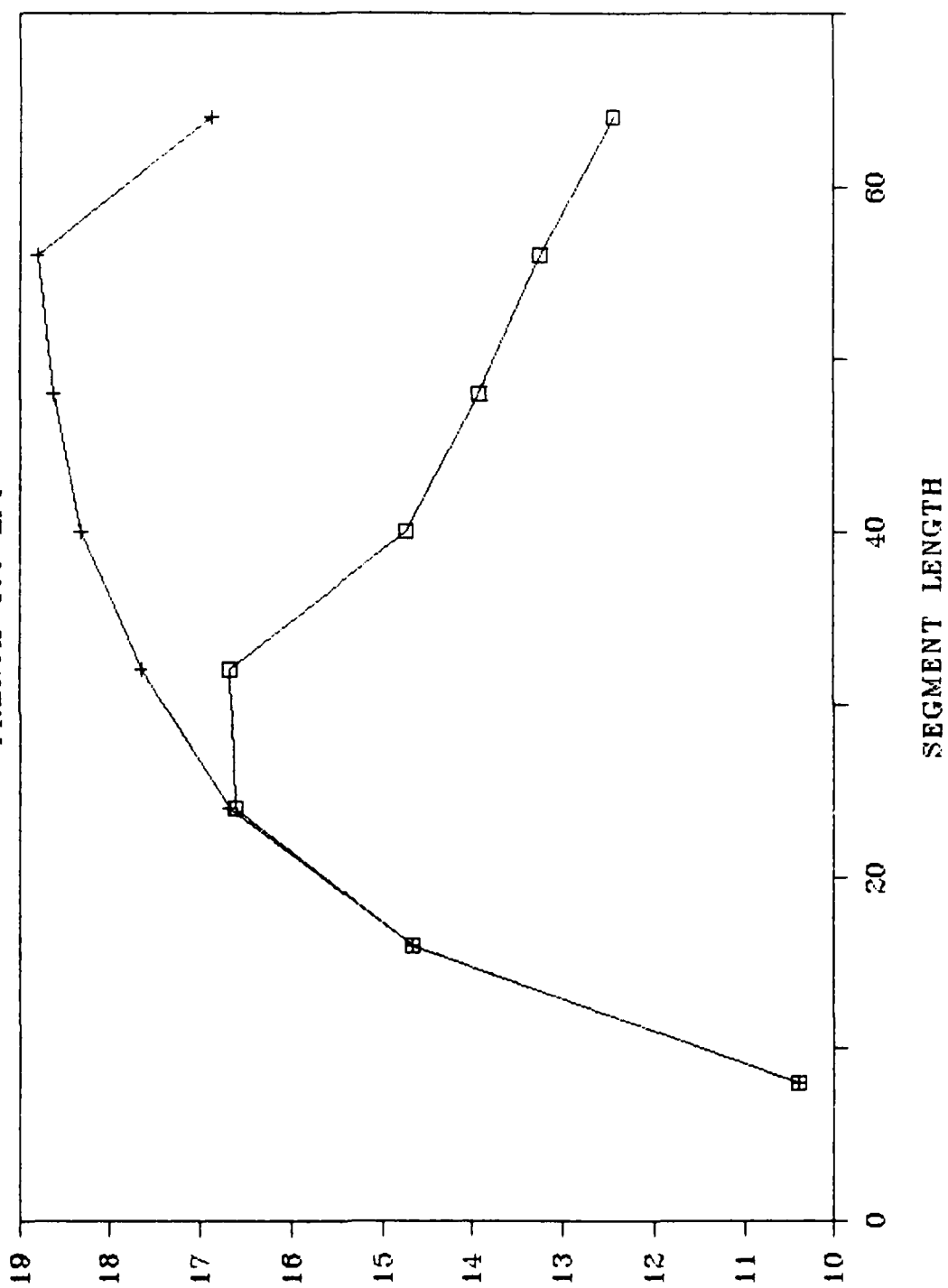
Figure 5.10

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	10.40	54.47
16	16	14.67	53.14
16	24	16.62	48.23
16	32	16.67	40.23
16	40	14.75	37.45
16	48	13.94	36.27
16	56	13.25	36.40
16	64	12.45	35.25
32	24	16.68	1:01.34
32	32	17.65	1:08.29
32	40	18.31	45.50
32	48	18.62	43.57
32	56	18.81	42.28
32	64	16.89	40.01

Table 5.11 - Compression Stats, French Journal - 400 LPI

SEGMENT LENGTH VS. COMPRESSION

FRENCH 400 LPI



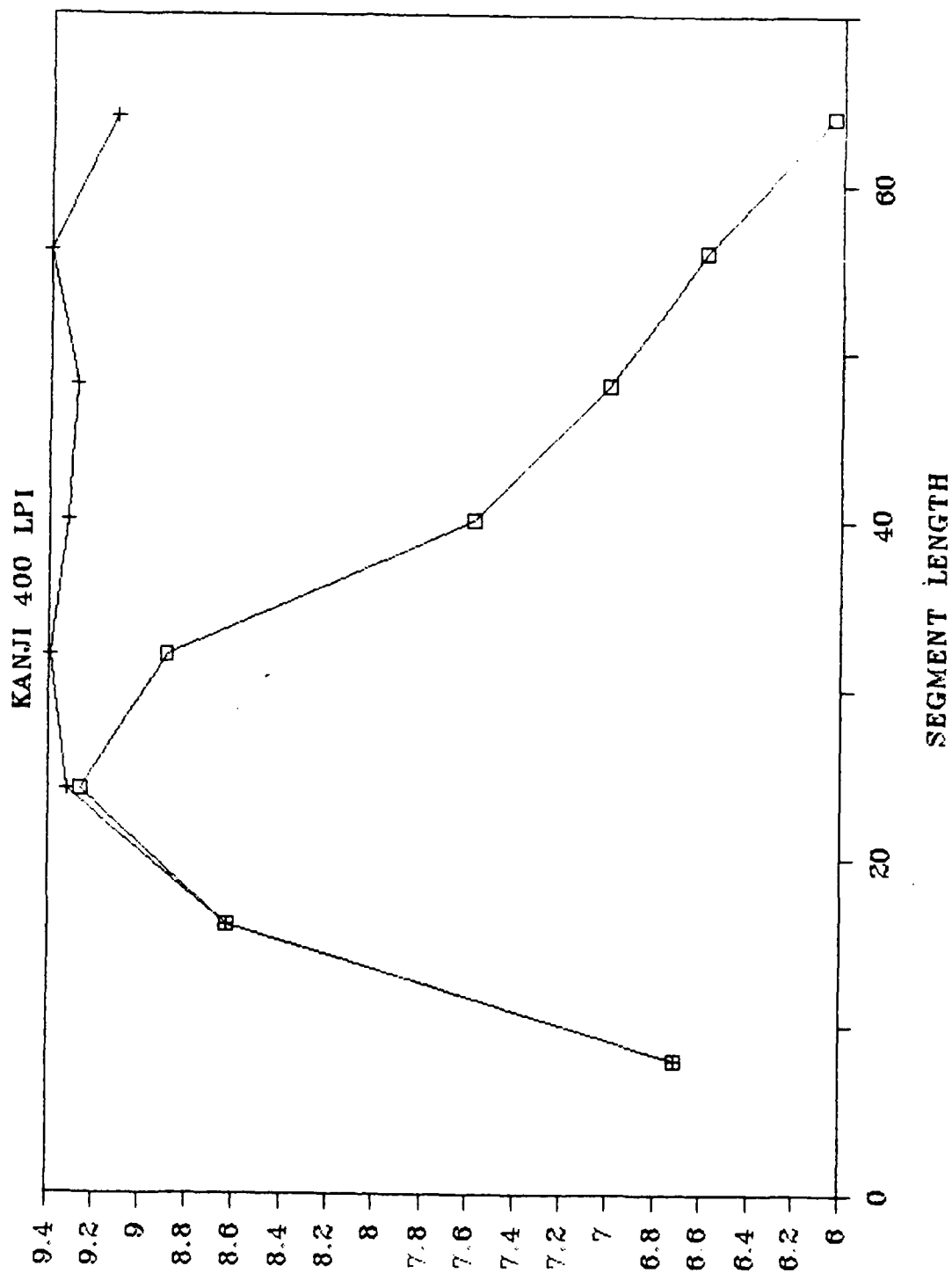
COMPRESSION

Figure 5.11

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	6.71	1:09:46
16	16	8.63	1:06.20
16	24	9.26	55.22
16	32	8.89	49.39
16	40	7.58	47.53
16	48	7.00	46.03
16	56	6.59	47.24
16	64	6.05	45.11
32	24	9.32	1:16.32
32	32	9.39	1:26.23
32	40	9.32	58.29
32	48	9.28	55.56
32	56	9.40	56.01
32	64	9.12	53.27

Table 5.12 - Compression Stats, Kanji Text - 400 LPI

SEGMENT LENGTH VS. COMPRESSION



COMPRESSION

Figure 5.12

Table 5.13 - Suggested Parameters for SLDC Encoding Program

Suggested Segment Lengths for Optimum SLDC Compression

<u>File</u>	<u>16 bit Arithmetic word length</u>	<u>32 bit Arithmetic word length</u>
English 200 lpi	24	40
French 200 lpi	24	40
Kanji 200 lpi	16	40
English 240 lpi	24	40
French 240 lpi	24	40
Kanji 240 lpi	16	40
English 300 lpi	24	48
French 300 lpi	24	48
Kanji 300 lpi	24	56
English 400 lpi	32	56
French 400 lpi	32	56
Kanji 400 lpi	24	56

An analysis of the 32 and 64 bit histograms described in the SOW versus those expected on the basis of a memoryless binomial distribution was made in order to determine if the deviation between the actual and expected probability distributions was significant. The results of this analysis are presented graphically in Figures 5.13 through 5.33 (In the probability distribution graphs, the \square indicates the expected distribution curve and the + indicates the actual distribution curve.). Every graph but the 200 resolution Kanji and 64 bit 400 resolution Kanji graphs have a second graph associated with it in which the deviation between the curves is presented in greater detail. The results show definite significant deviation between the actual and expected histograms, and is most evident in the Kanji graphs. It appears that the number of occurrences in both 32 and 64 bit cases of segment weights < 3 are significantly lower than expected, and occurrences of segment weights > 3 are significantly higher than expected. Attempts were made to alleviate this problem by trying to produce a more predictable CIF by increasing the neighbor template from 4 to 7 bits. The three extra bits were placed in different positions available to the decoder, around the already existing template. This attempt made the actual curve slightly more binomial in nature, but changes in compression were small.

There are two possible ways to improve the SLDC technique. The first is the aforementioned attempt to improve the CIF to be more binomially predictable, which was not successful. The second would be to replace the predictive encoding algorithm, which is binomially based, with a function which better fits the actual statistics of the CIF's.

ACTUAL VS. EXPECTED HISTOGRAM

ENGLISH 200 LPI 32 BIT SEGMENT

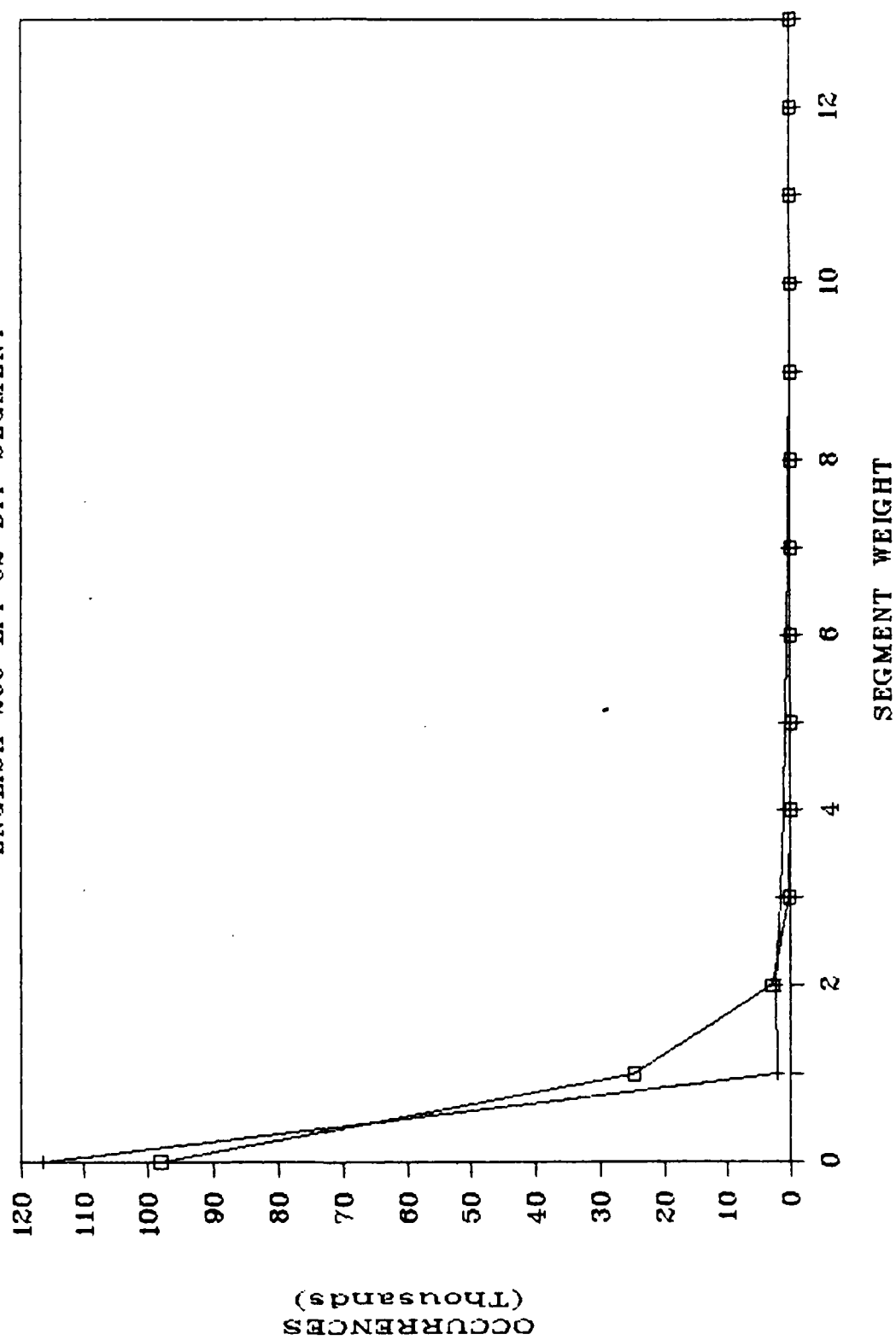


Figure 5.13 - Full Scale

ACTUAL VS. EXPECTED HISTOGRAM

ENGLISH 200 LPI 32 BIT SEGMENT

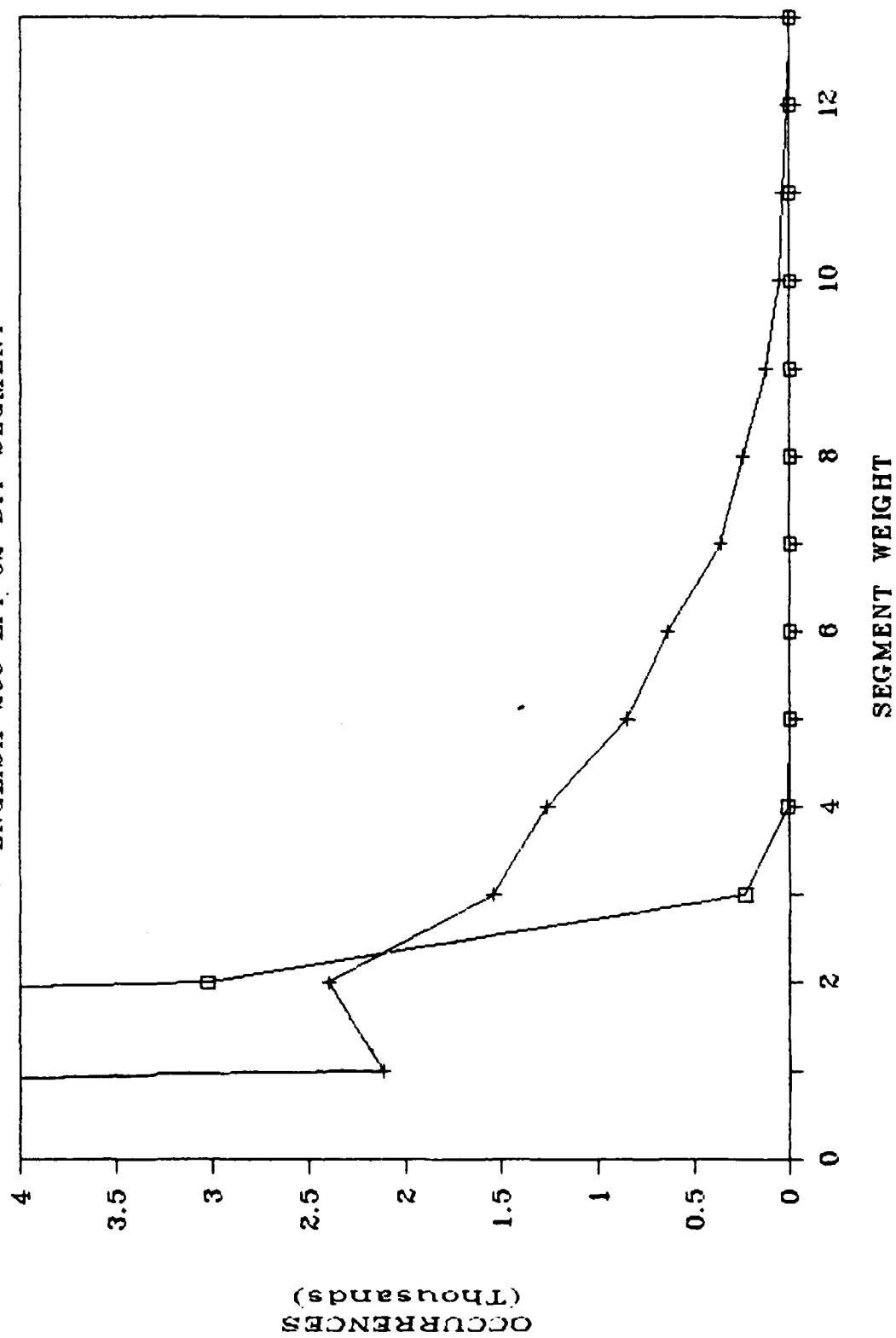


Figure 5.14 - Enlarged Section

ACTUAL VS. EXPECTED HISTOGRAM

ENGLISH 200 LPI 64 BIT SEGMENT

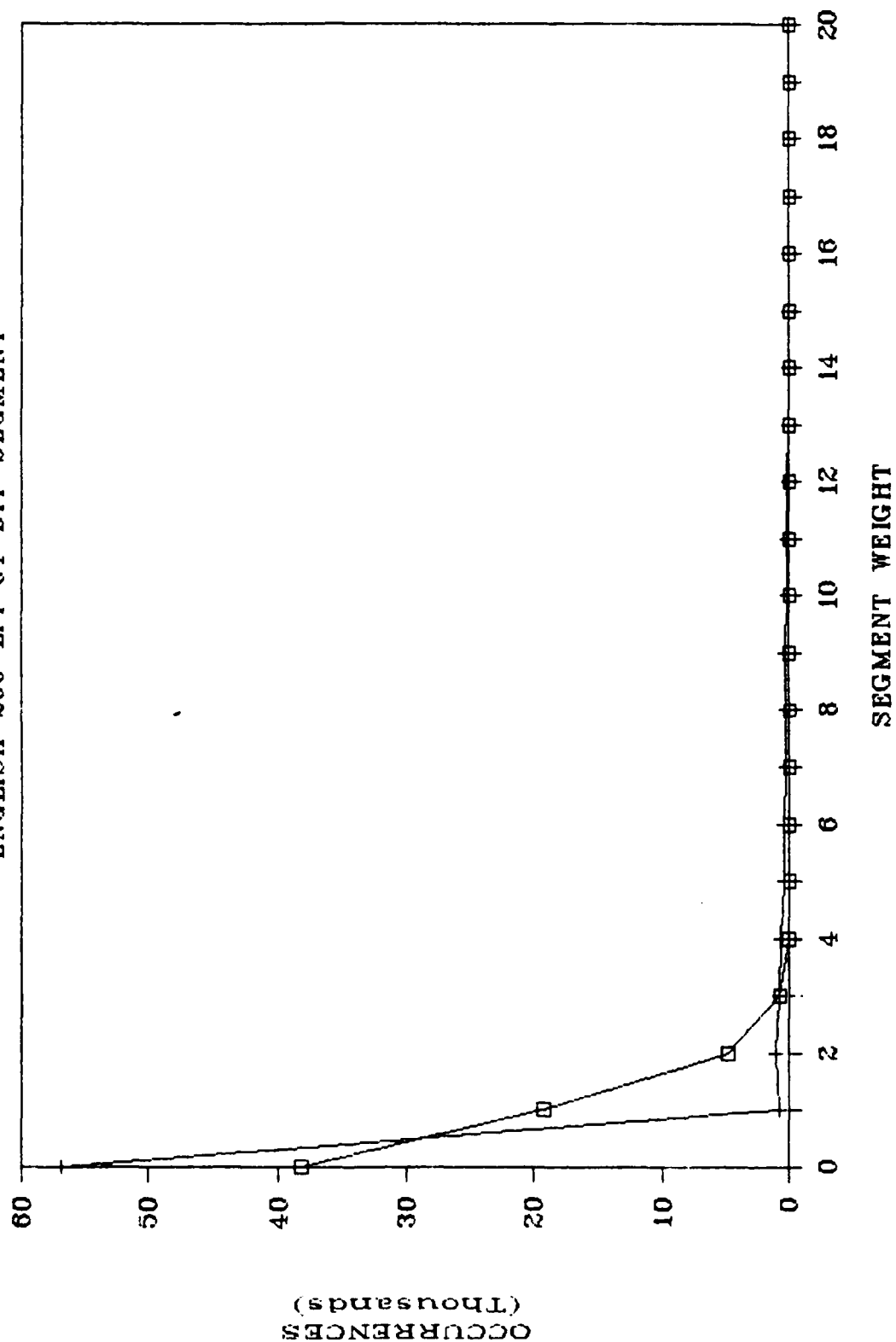


Figure 5.15 - Full Scale

ACTUAL VS. EXPECTED HISTOGRAM ENGLISH 200 LPI 64 BIT SEGMENT

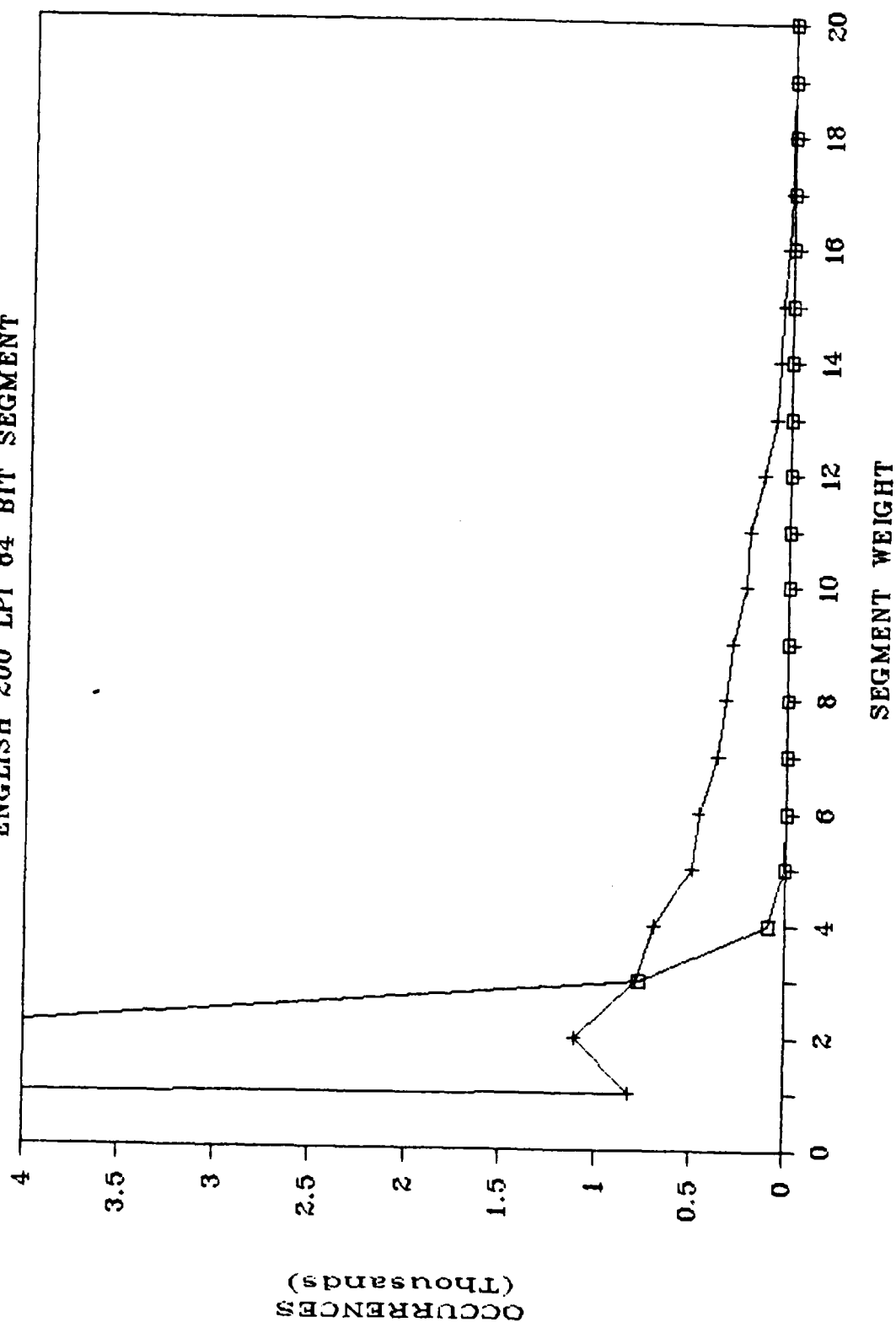


Figure 5.16 - Enlarged Section

ACTUAL VS. EXPECTED HISTOGRAM

FRENCH 200 LPI 32 BIT SEGMENT

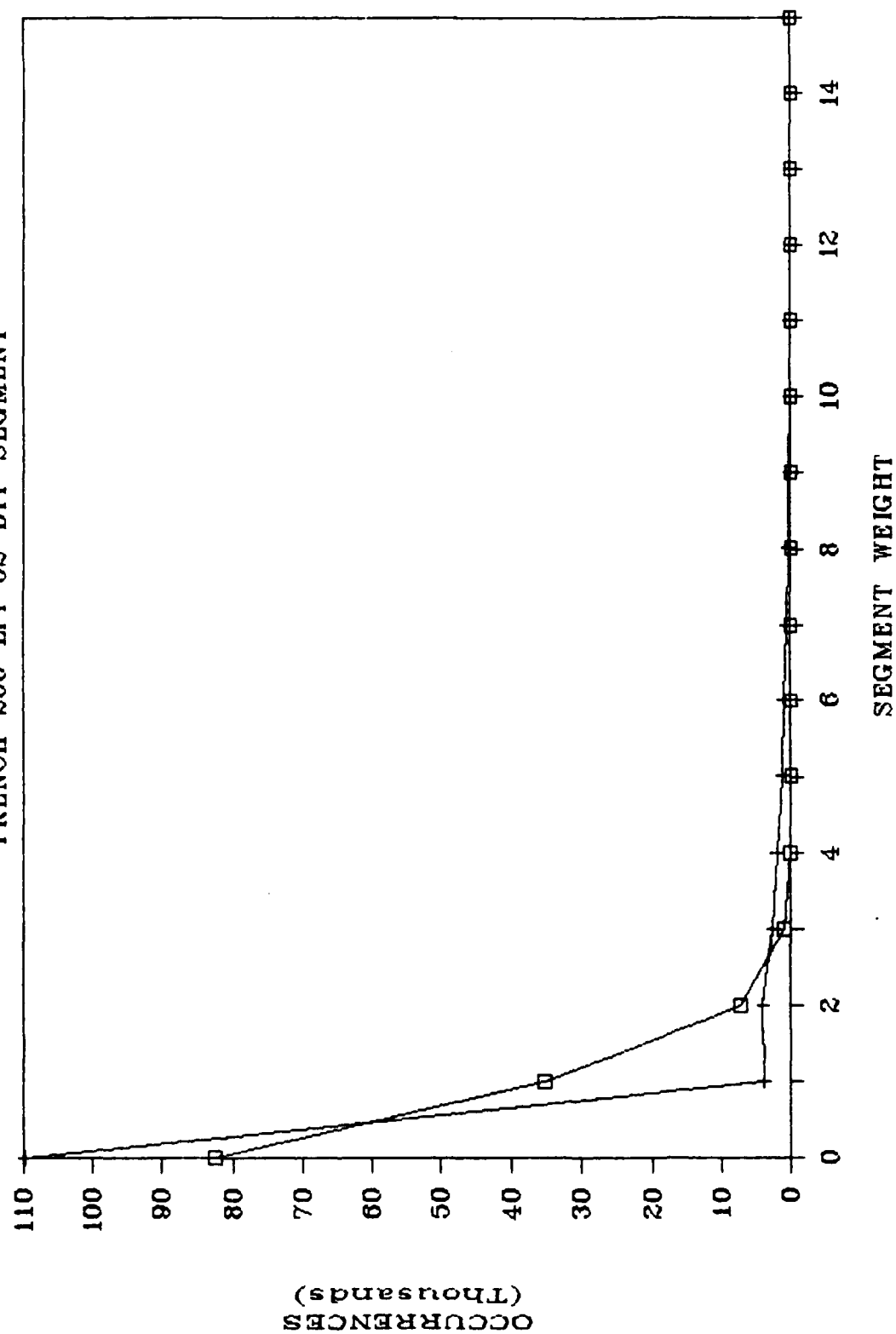


Figure 5.17 - Full Scale

ACTUAL VS. EXPECTED HISTOGRAM

FRENCH 200 LPI 32 BIT SEGMENT

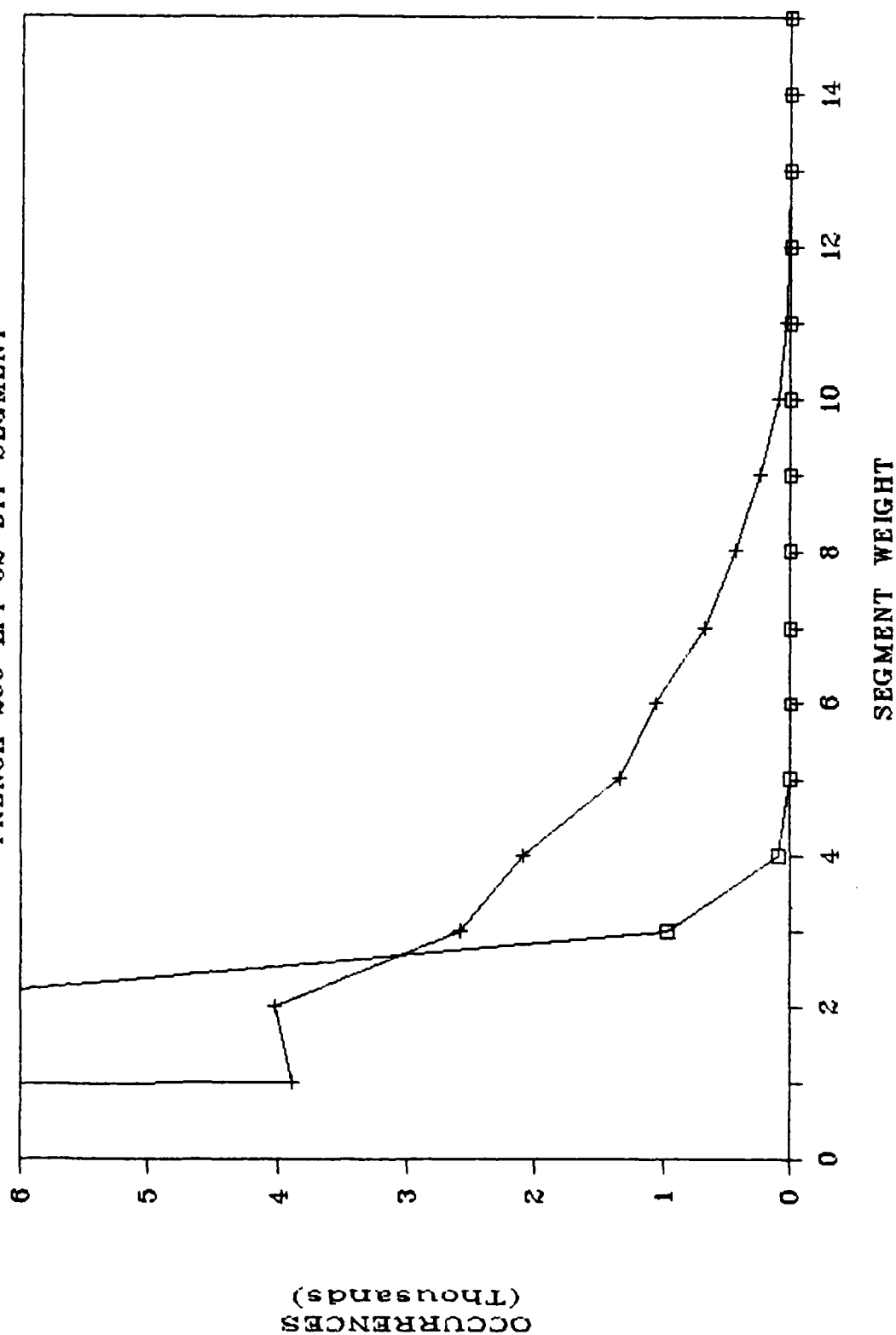


Figure 5.18 - Enlarged Section

ACTUAL VS. EXPECTED HISTOGRAM

FRENCH 200 LPI 64 BIT SEGMENT

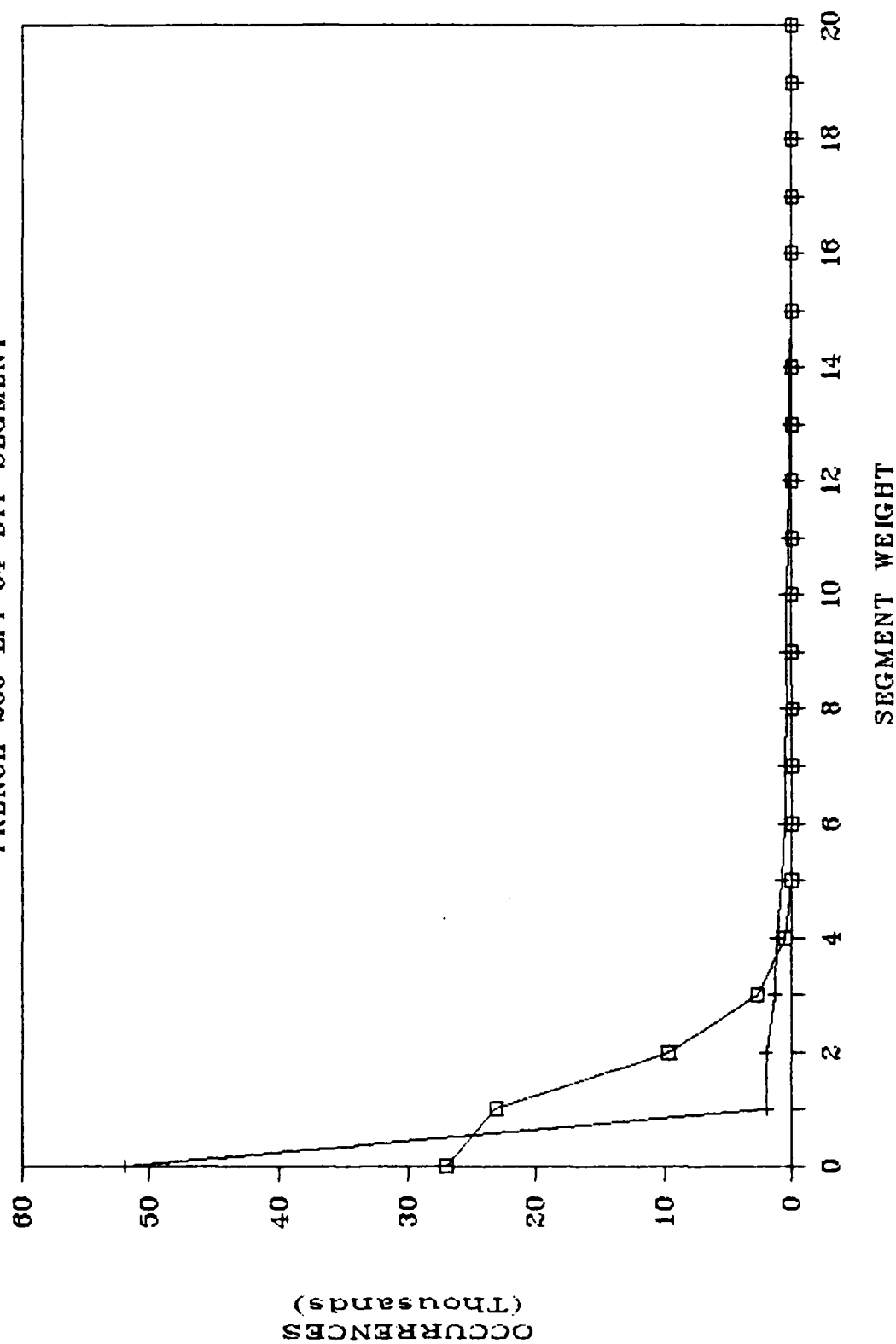


Figure 5.19 - Full Scale

ACTUAL VS. EXPECTED HISTOGRAM

FRENCH 200 LPI 64 BIT SEGMENT

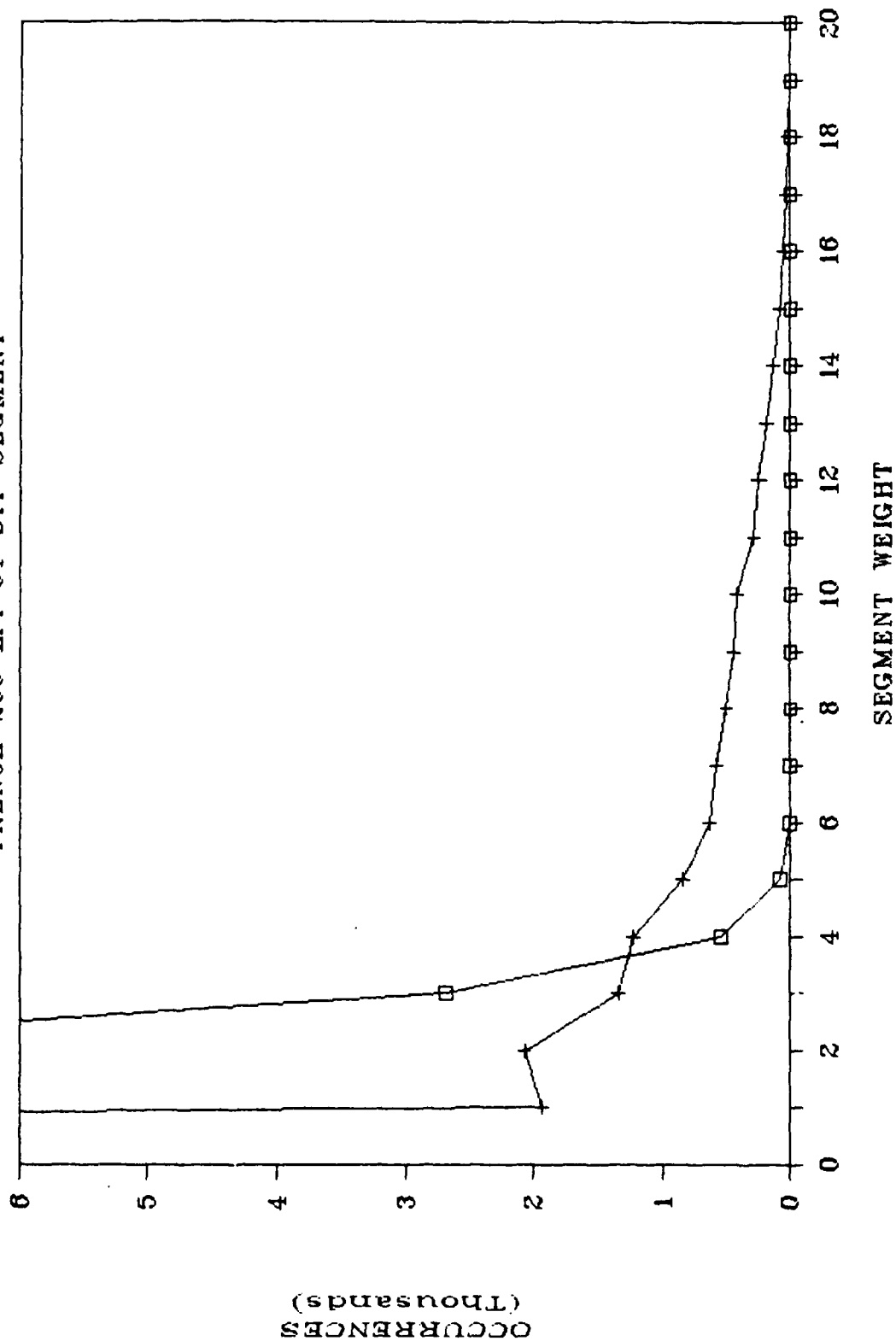


Figure 5.20 - Enlarged Section

ACTUAL VS. EXPECTED HISTOGRAM

KANJI 200 LPI 32 BIT SEGMENT

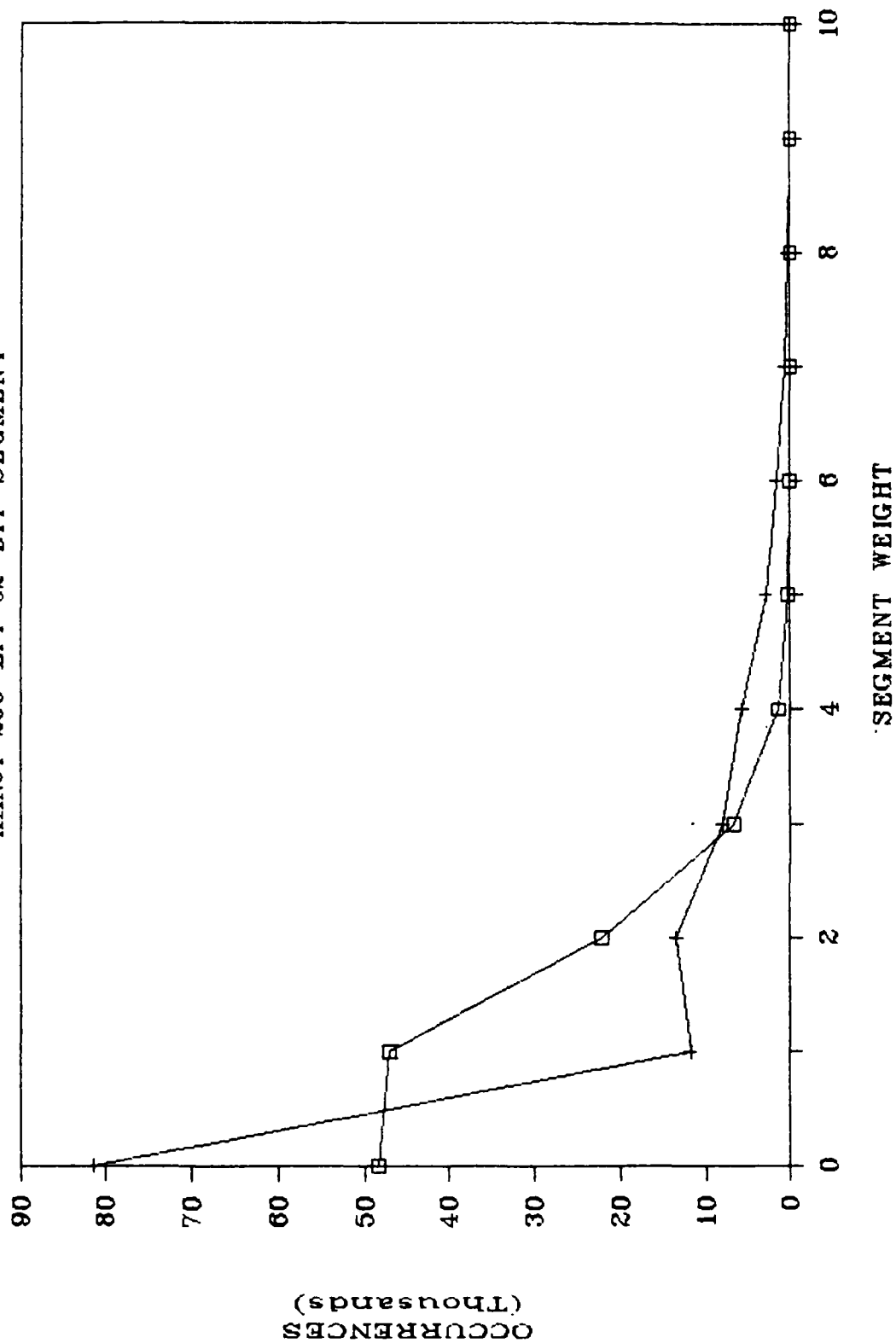


Figure 5.21 - Full Scale

ACTUAL vs. EXPECTED HISTOGRAM

KANJI 200 LPI 64 BIT SEGMENT

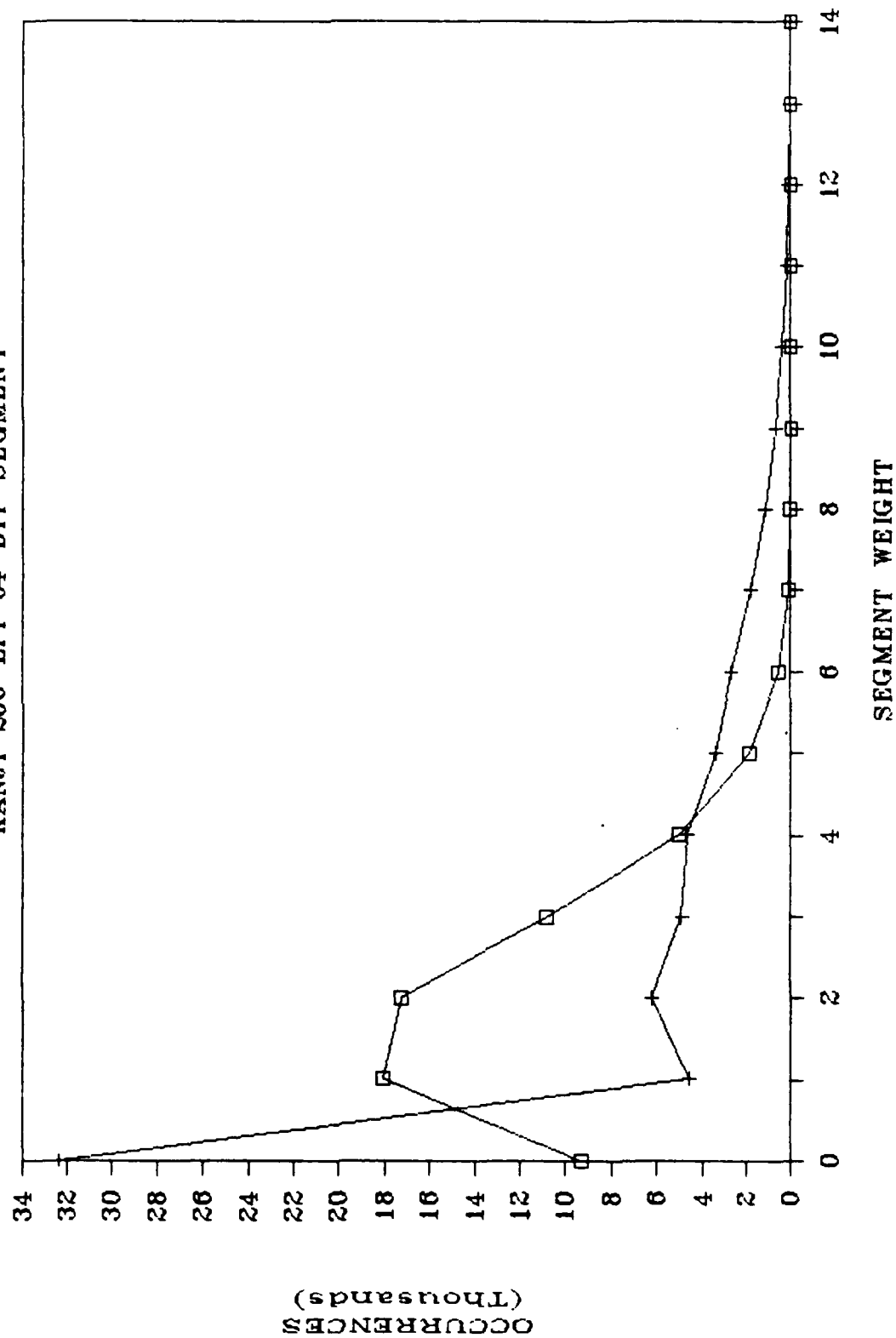


Figure 5.22 - Full Scale

ACTUAL vs. EXPECTED HISTOGRAM

ENGLISH 400 LPI 32 BIT SEGMENT

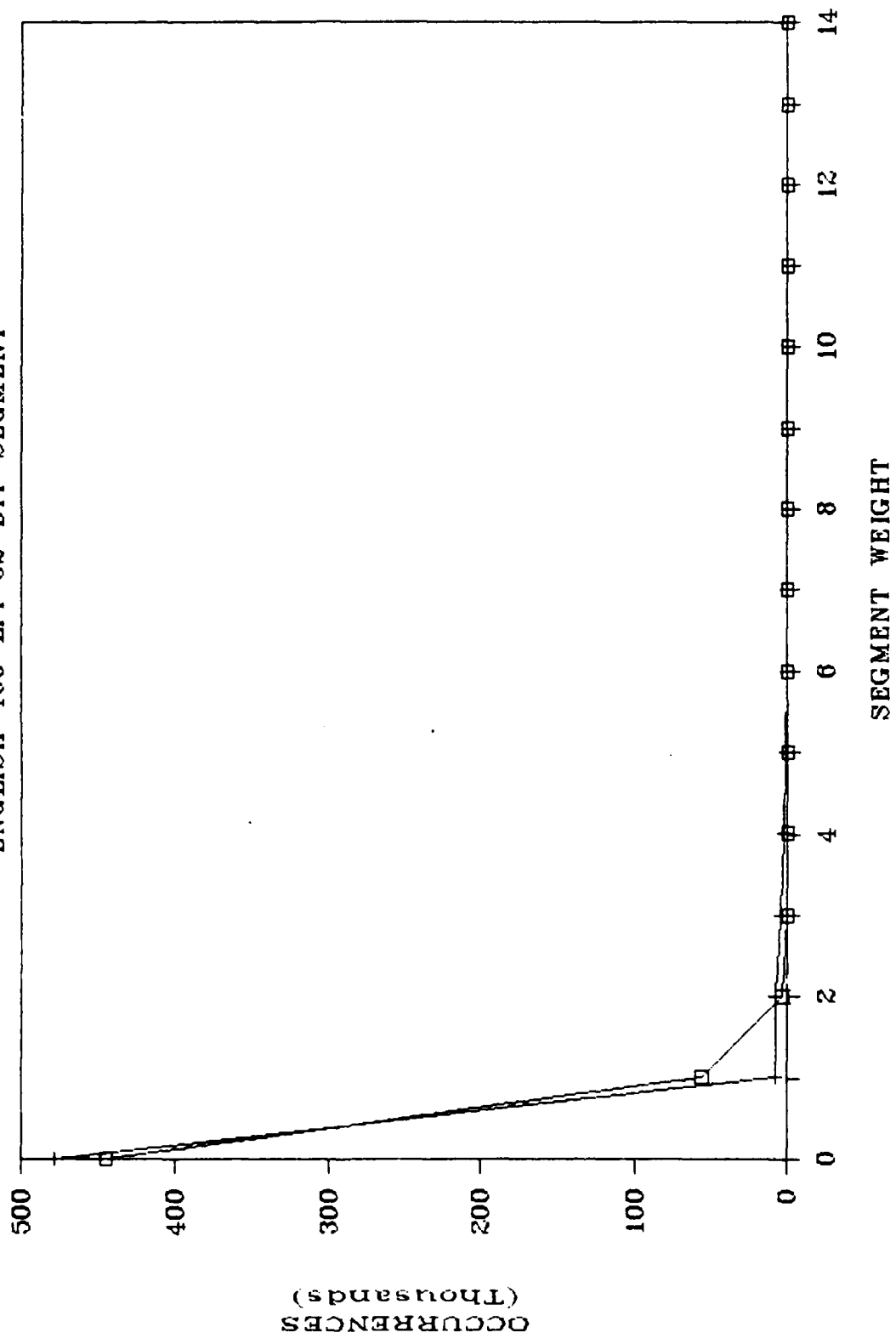


Figure 5.23 - Full Scale

ACTUAL VS. EXPECTED HISTOGRAM

ENGLISH 400 LPI 32 BIT SEGMENT

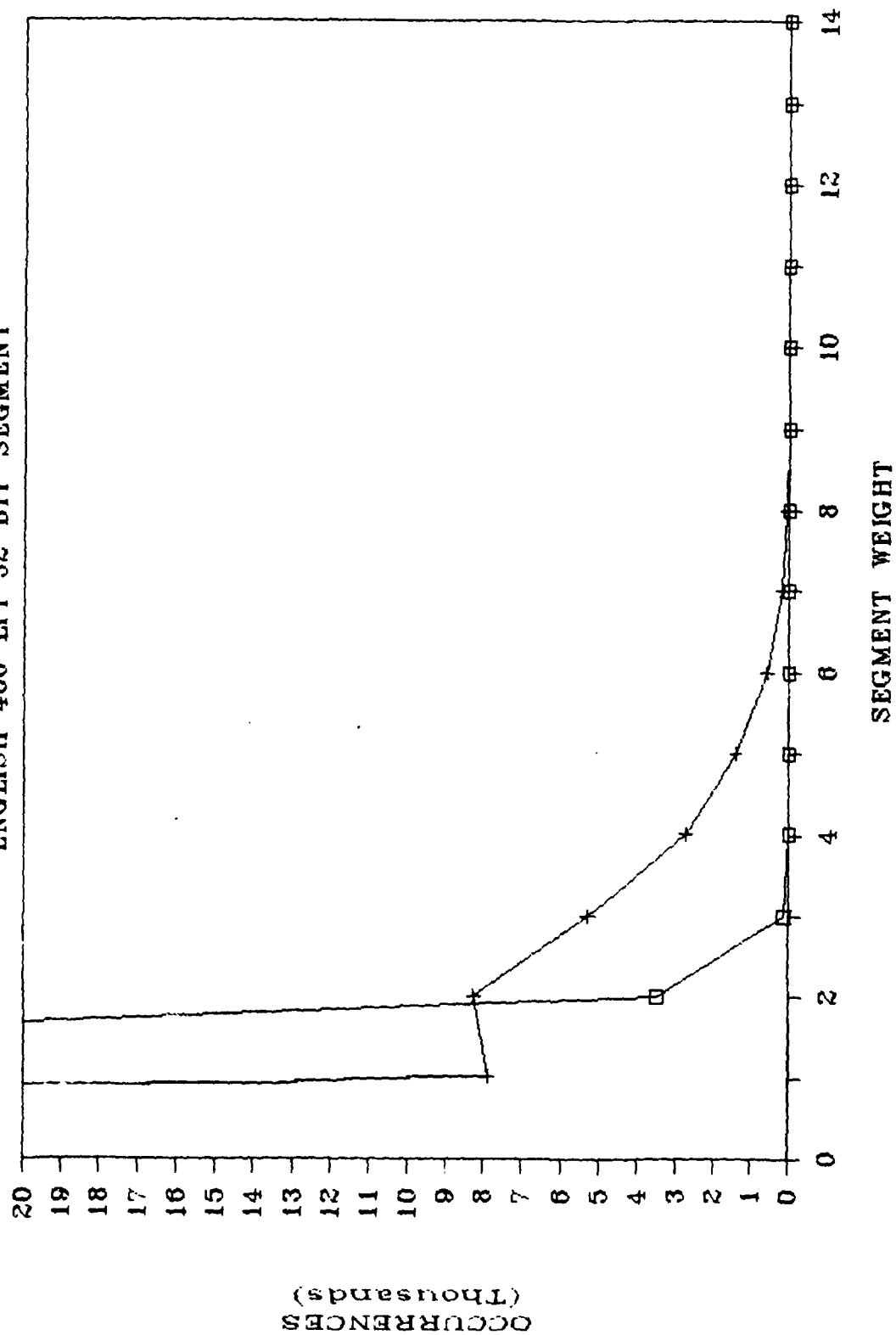


Figure 5.24 - Enlarged Section

ACTUAL VS. EXPECTED HISTOGRAM

ENGLISH 400 LPI 64 BIT SEGMENT

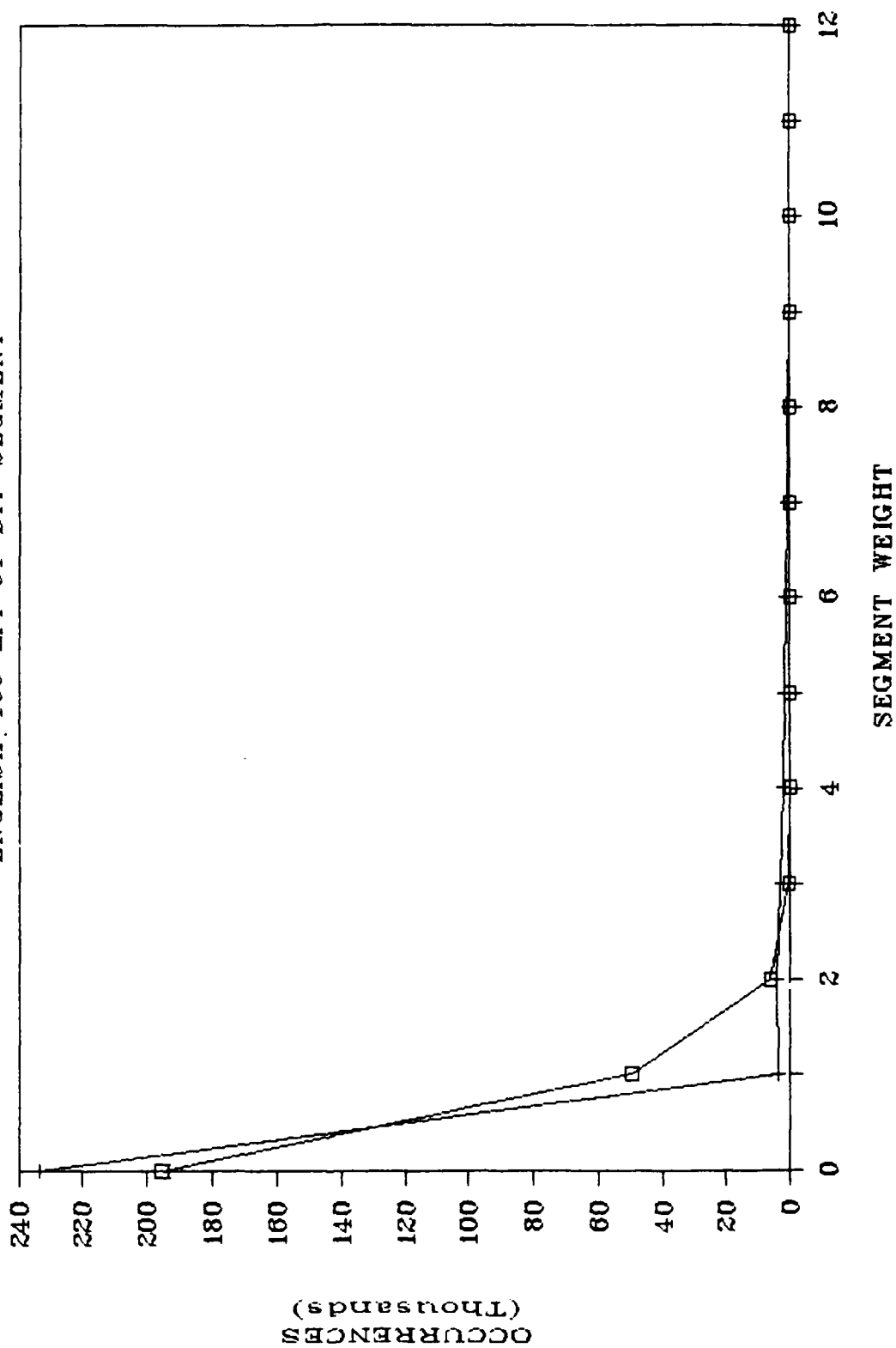


Figure 5.25 - Full Scale

ACTUAL VS. EXPECTED HISTOGRAM

ENGLISH 400 LPI 64 BIT SEGMENT

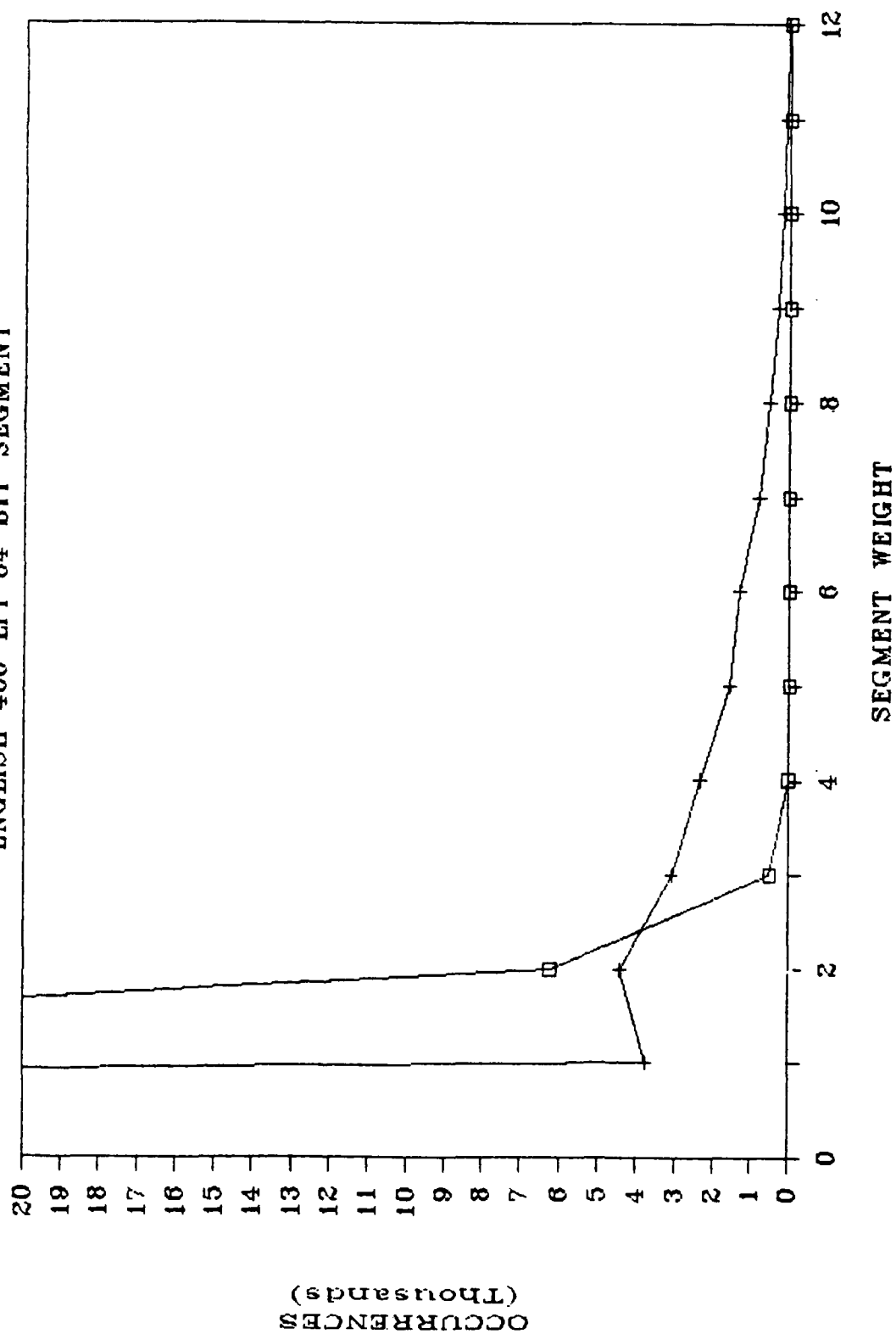


Figure 5.26 - Enlarged Section

ACTUAL VS. EXPECTED HISTOGRAM

FRENCH 400 LPI 32 BIT SEGMENT

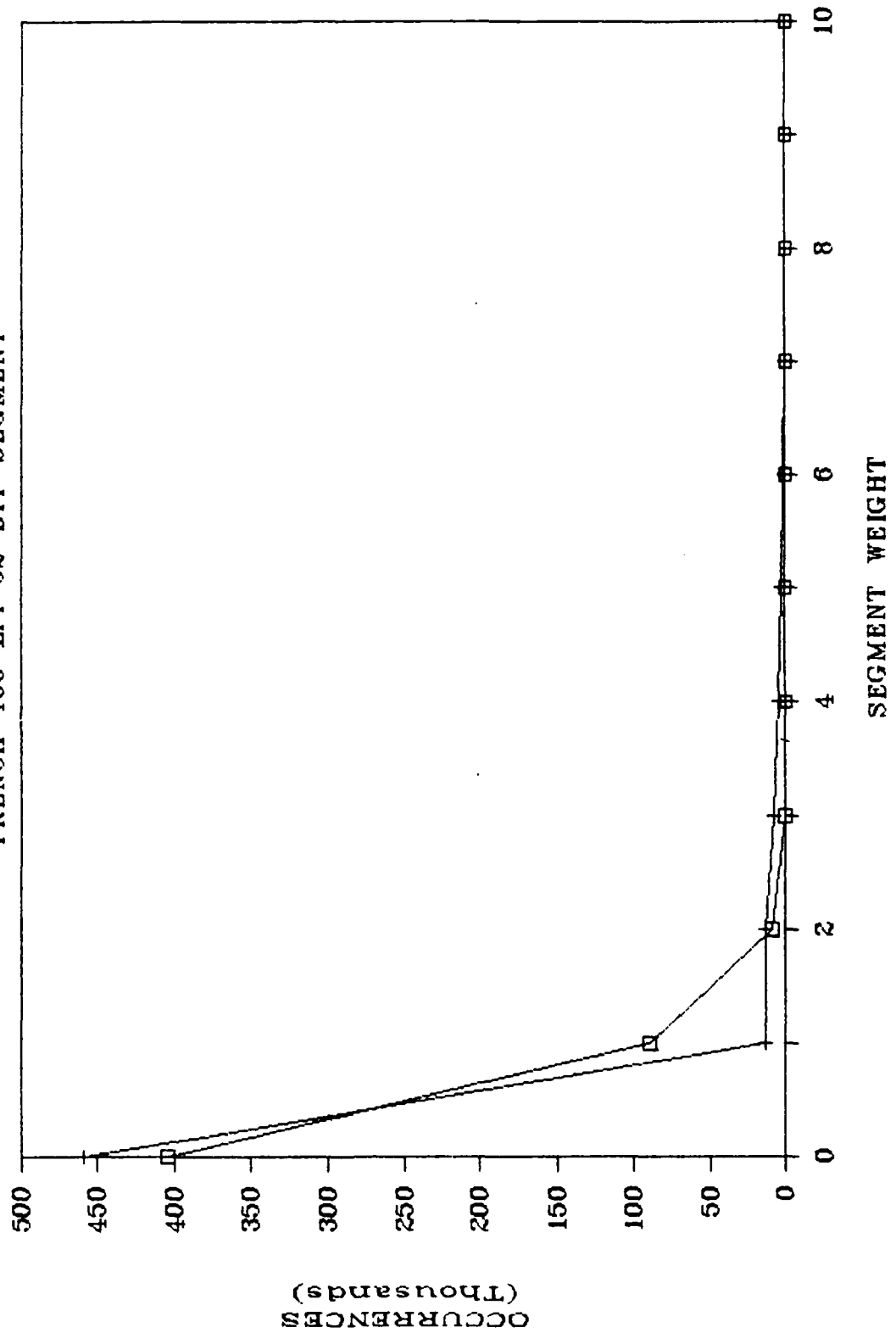


Figure 5.27 - Full Scale

ACTUAL VS. EXPECTED HISTOGRAM

FRENCH 400 LPI 32 BIT SEGMENT

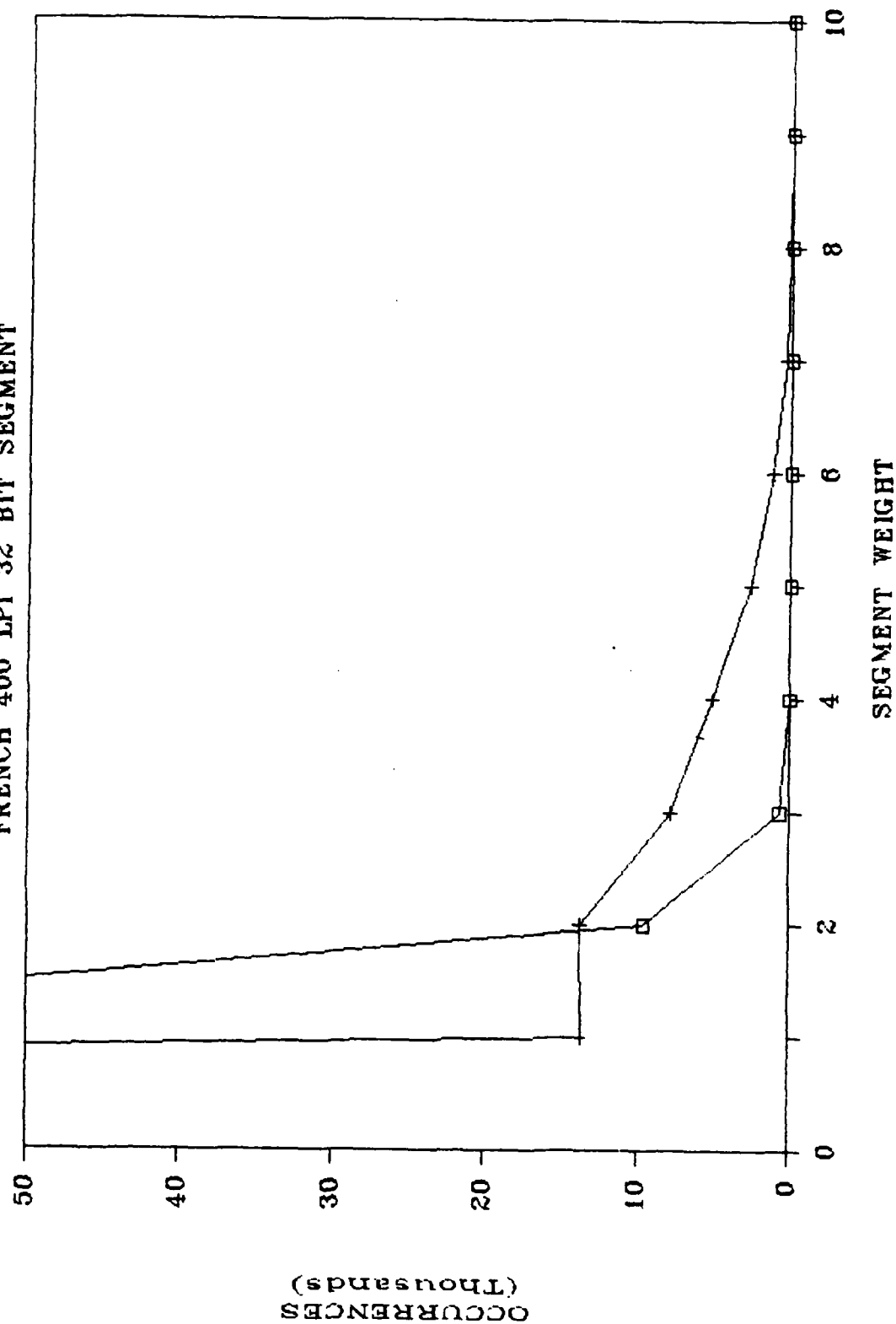


Figure 5.28 - Enlarged Section

ACTUAL VS. EXPECTED HISTOGRAM

FRENCH 400 LPI 64 BIT SEGMENT

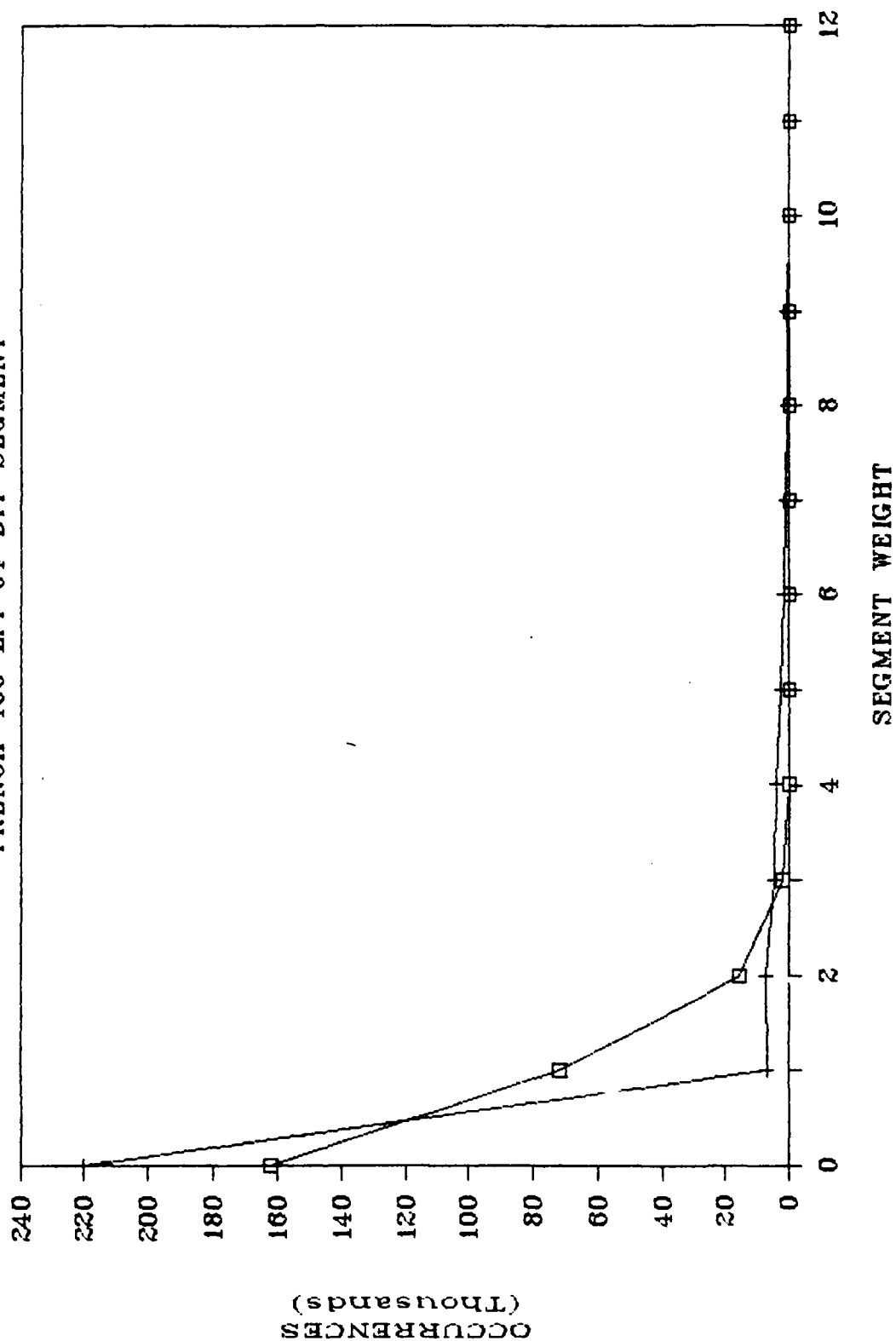


Figure 5.29 - Full Scale

ACTUAL VS. EXPECTED HISTOGRAM

FRENCH 400 LPI 64 BIT SEGMENT

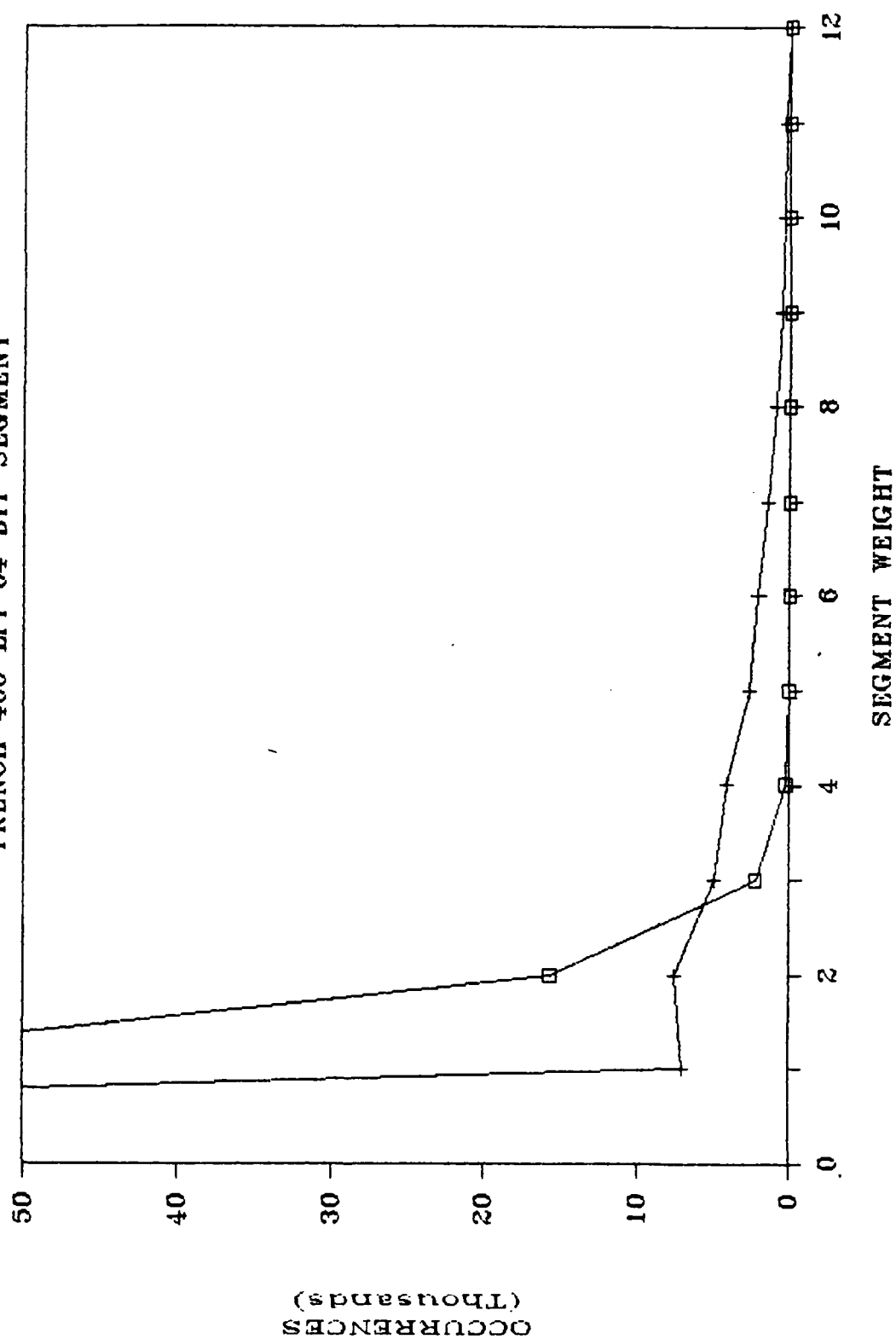


Figure 5.30 - Enlarged Section

ACTUAL VS. EXPECTED HISTOGRAM

KANJI 400 LPI 32 BIT SEGMENT

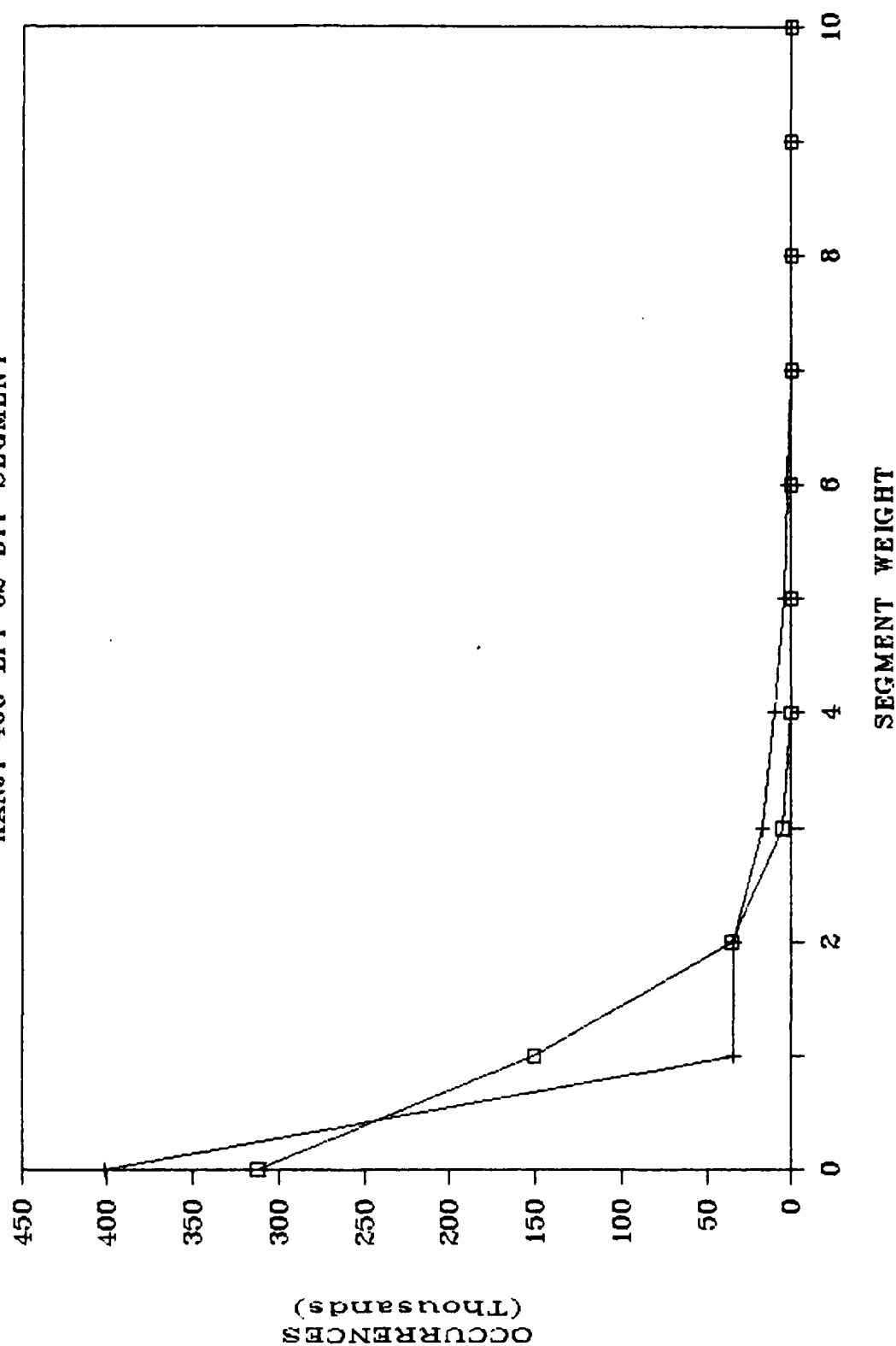


Figure 5.31 - Full Scale

ACTUAL VS. EXPECTED HISTOGRAM

KANJI 400 LPI 32 BIT SEGMENT

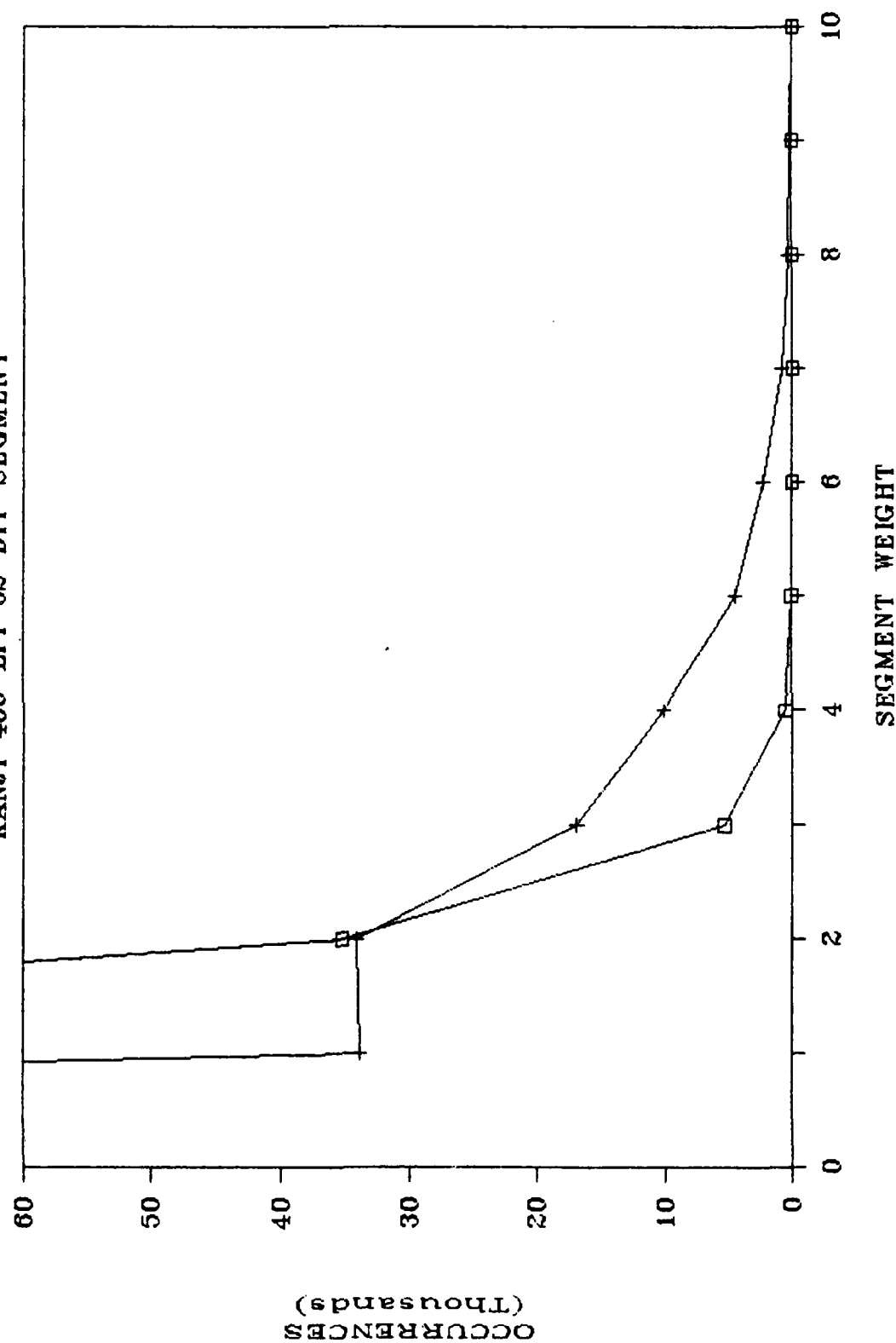


Figure 5.32 - Enlarged Section

ACTUAL vs. EXPECTED HISTOGRAM

KANJI 400 LPI 64 BIT SEGMENT

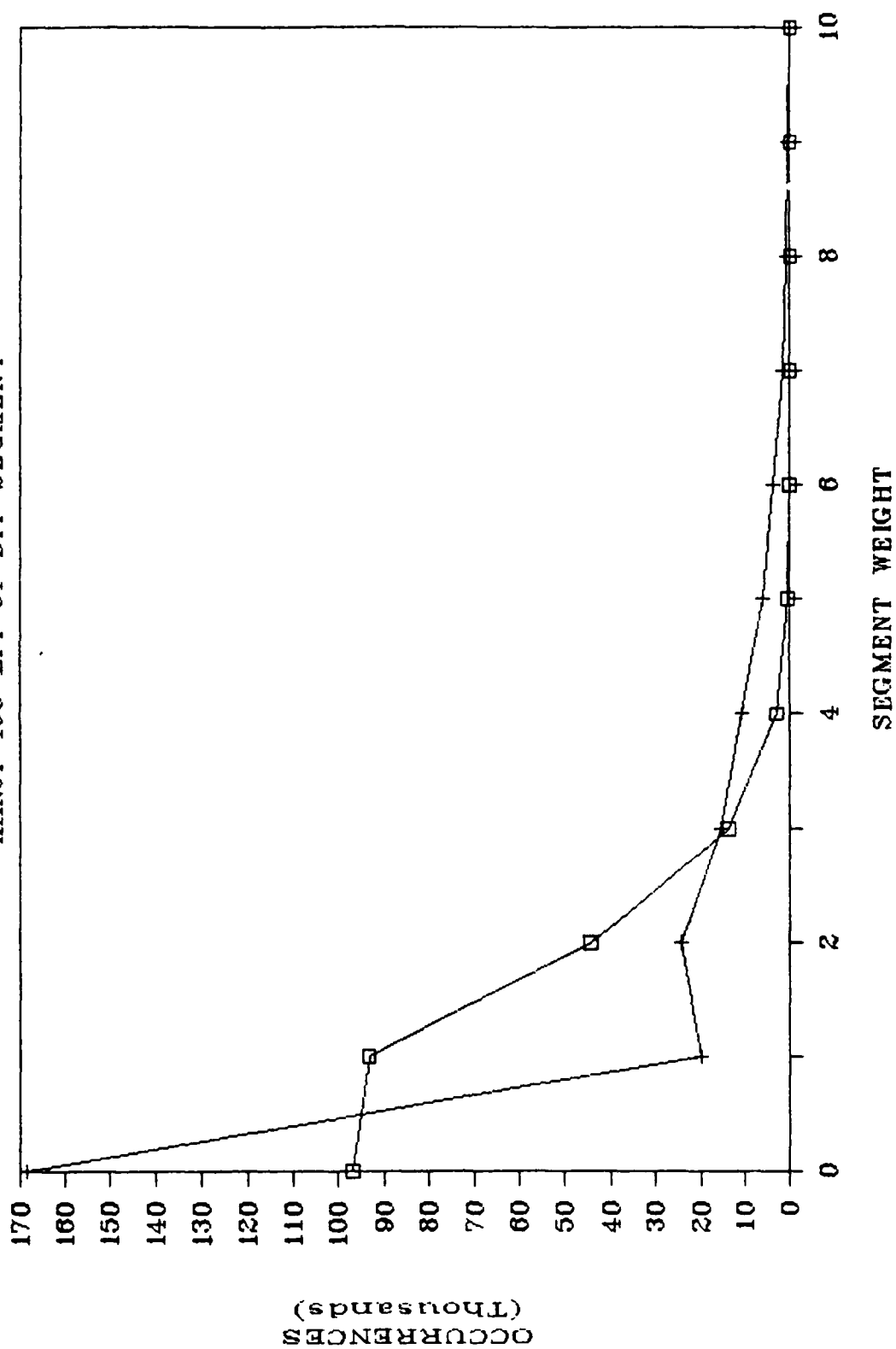


Figure 5.33 - Full Scale

6.0 Conclusions and Recommendations

In simulation runs of the SLDC algorithm, the overall compression achieved by compressing CIF's using the SLDC encoding algorithm did not compare favorably with the higher compression achieved by the MODREAD II algorithm run on the same original images. A side-by-side comparison of the results of the two images is presented in Table 6.1; the large difference in compression between the two algorithms is made more evident when displayed graphically (Figures 6.1 through 6.4; in the comparison graphs, the \square indicates the SLDC compression results and the + indicates the MODREAD II compression results.).

Compression differences between individual images show the Kanji documents to have the closest compression to MODREAD II, followed by the English, then French documents. The difference in compression between the two algorithms increases as the resolution increases. The compression vs. resolution curves of both algorithms are linear, with slope differences varying by a factor of two on the average, with MODREAD II having the steeper, more positive slope.

A possible suggestion to reach better compression, closer to MODREAD II, would be a better predicting function, as mentioned earlier, to better match the actual statistics of the CIF's. Another suggestion to consider is a new CIF prediction scheme which would generate CIF statistics that would more closely match those of a binomial distribution.

CCITT IMAGE	Resolution, LPI	SLDC Compression Ratio	MODREAD II Compression Ratio	% Diff
#1	200	20.17	30.57	34.0
English	240	23.16	36.54	36.6
Letter	300	27.43	45.44	39.6
	400	33.45	59.57	43.8
#5	200	11.61	17.61	34.1
French	240	12.90	21.00	38.6
Journal	300	15.10	25.91	41.7
	400	18.81	34.55	45.5
#7	200	5.44	7.59	28.3
KANJI	240	6.23	9.12	31.7
TEXT	300	7.34	11.43	35.8
	400	9.40	15.50	39.4
AVERAGE	200	9.39	13.56	30.8
	240	10.67	16.25	34.3
	300	12.56	20.26	38.0
	400	15.84	27.22	41.8

Table 6.1 - SLDC vs. MODREAD II Comparison

SLDC vs. MODREAD II ENGLISH LETTER

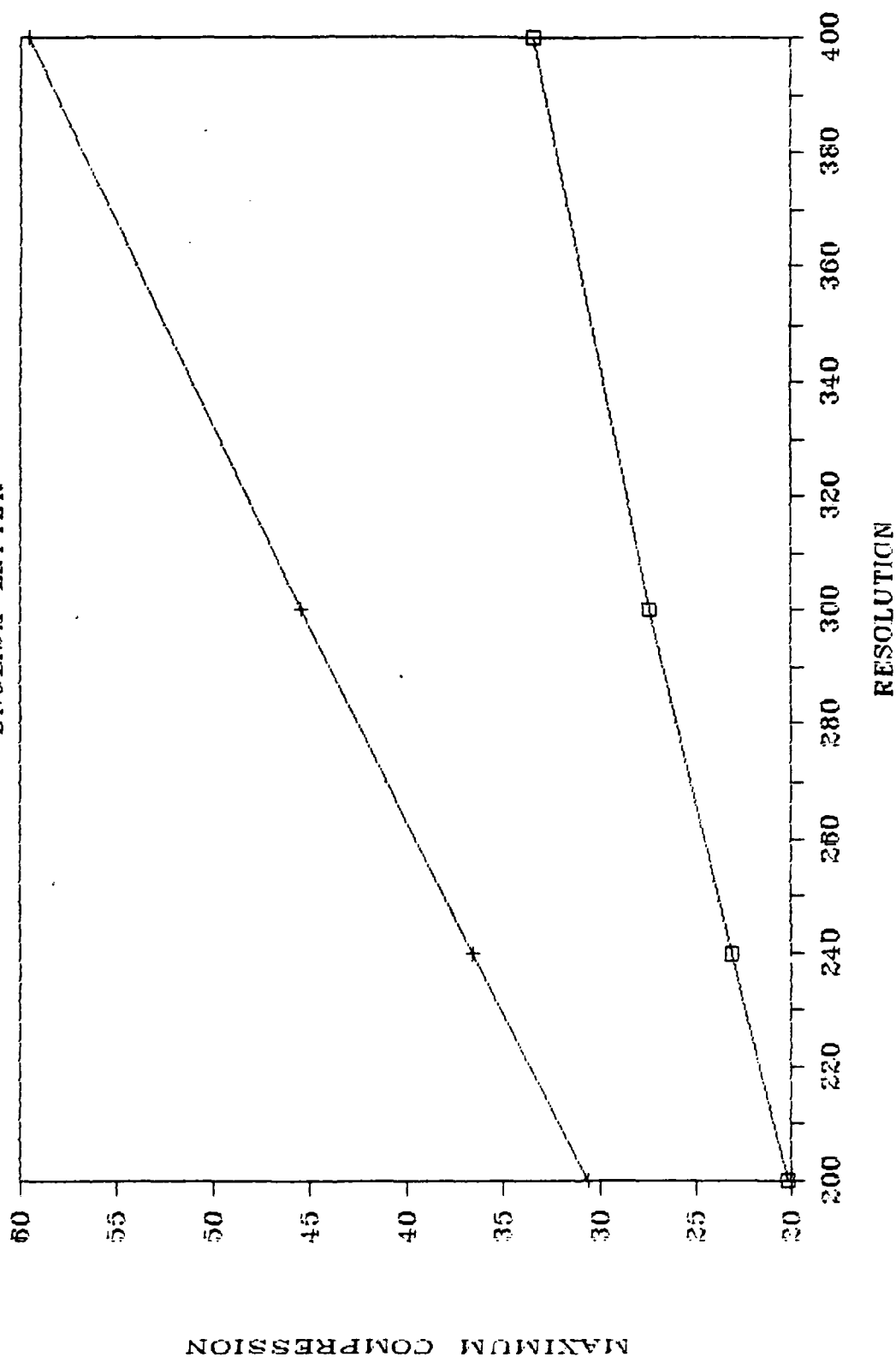


Figure 6.1

SLDC vs. MODREAD II

FRENCH JOURNAL

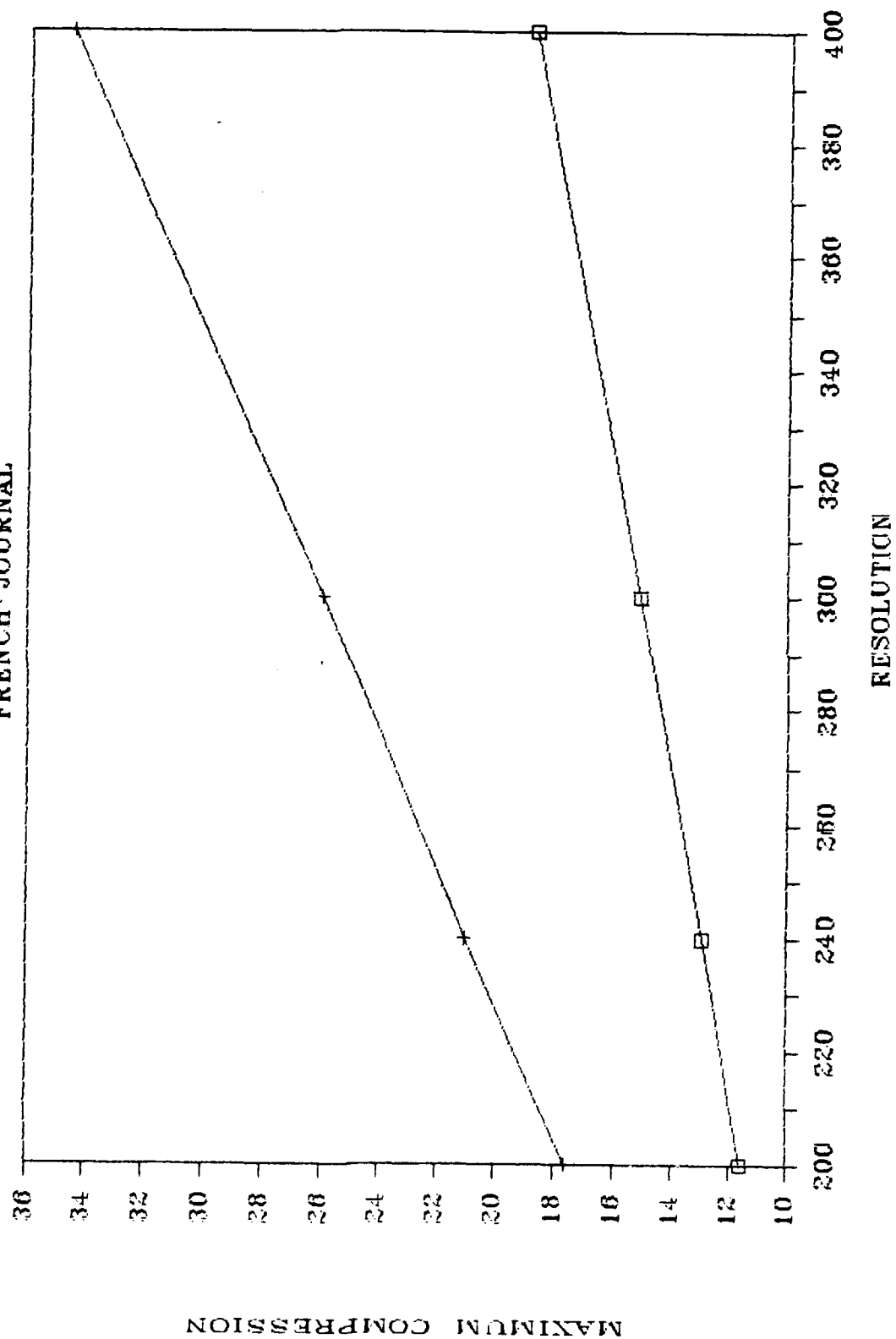


Figure 6.2

AD-A160 846

SCAN LINE DIFFERENCE COMPRESSION ALGORITHM SIMULATION
STUDY(U) DELTA INFORMATION SYSTEMS INC MORSHAM PA
AUG 85 NCS-T10-85-6 DCA100-83-C-0047

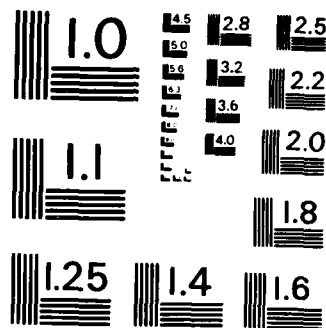
2/2

UNCLASSIFIED

F/G 9/2

NL

										END			
										FILED			
										DTIC			



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

SLDC vs. MODREAD II

KANJI TEXT

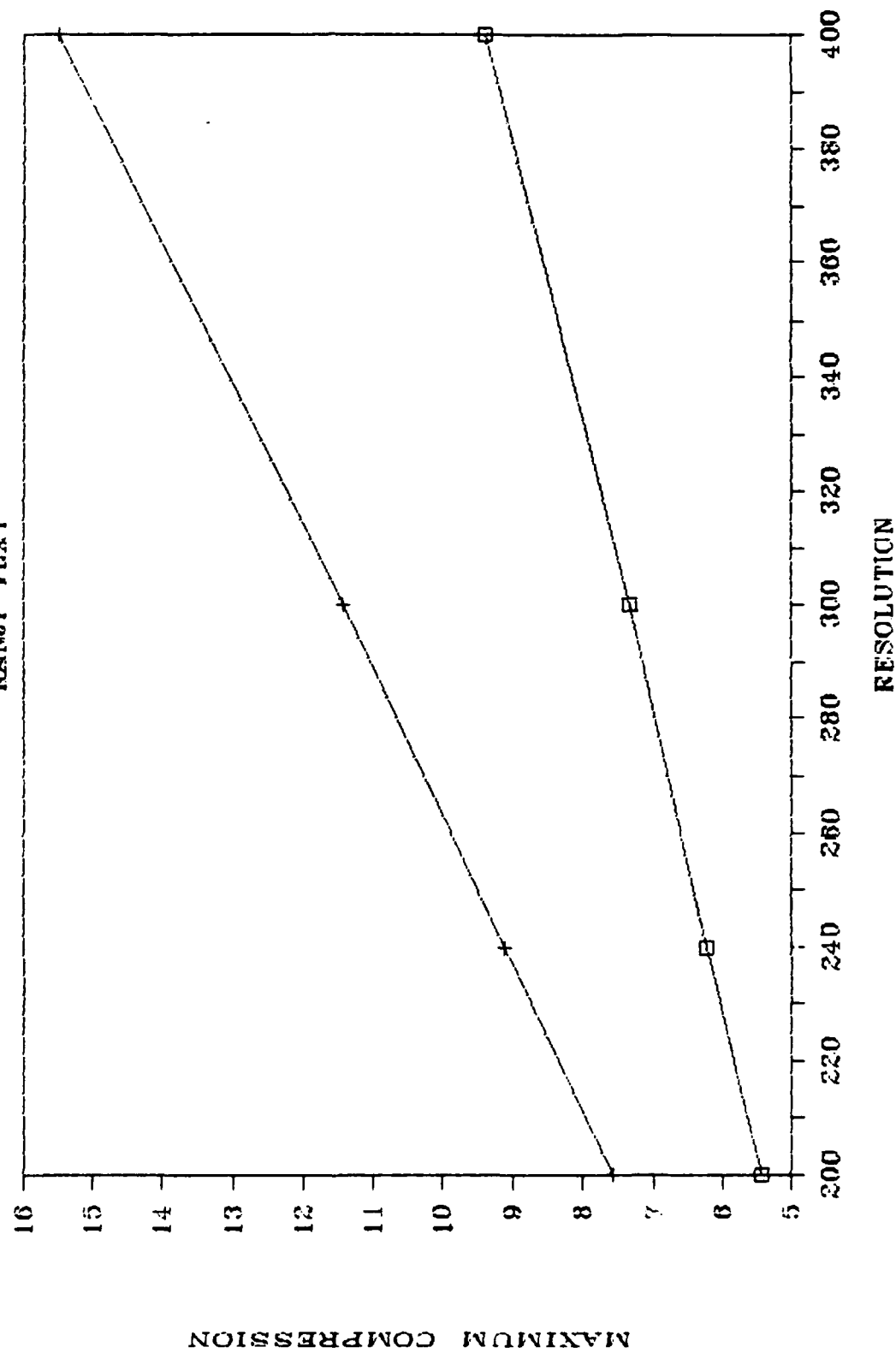


Figure 6.3

SLDC vs. MODREAD II

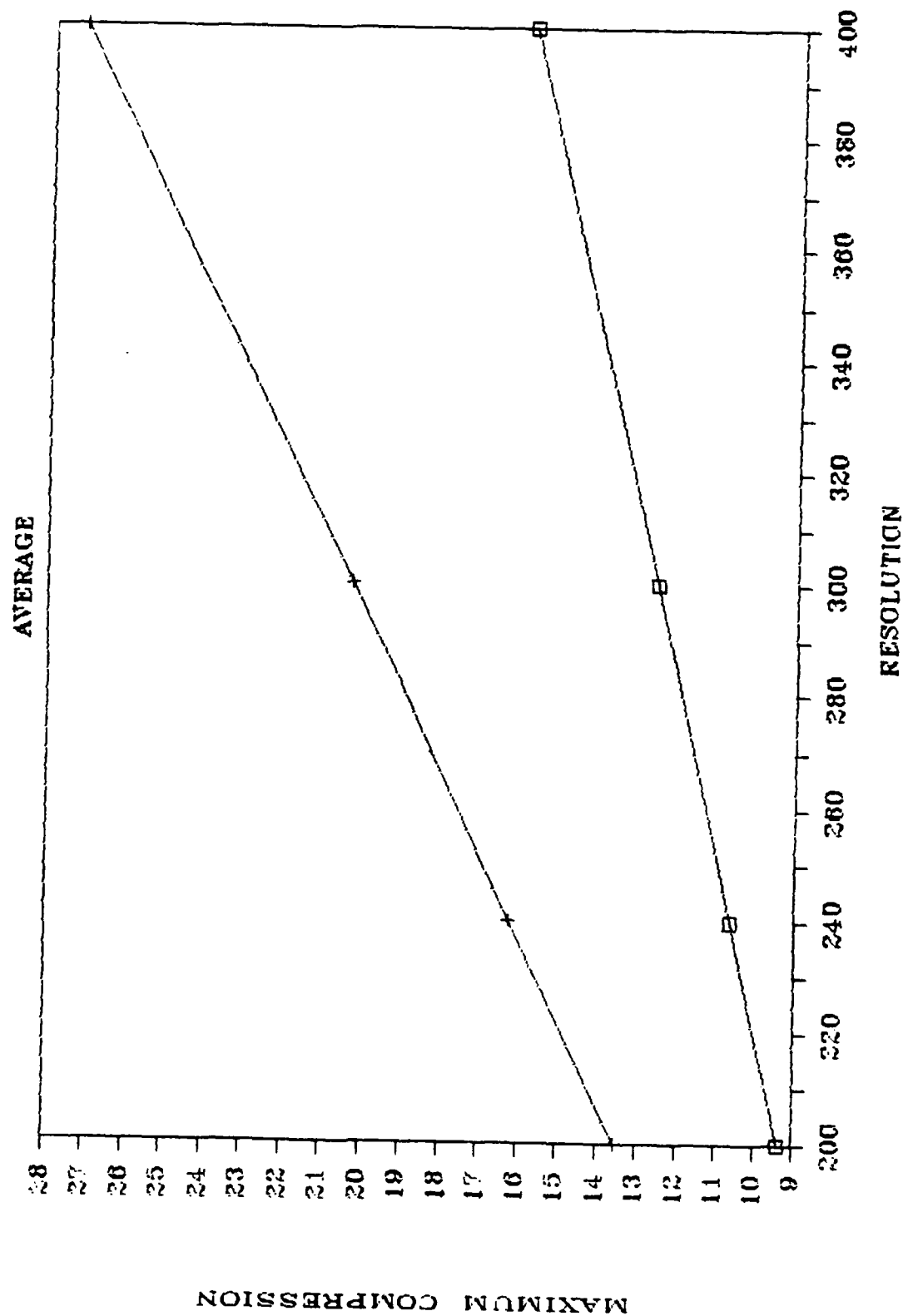


Figure 6.4

APPENDIX A

DESCRIPTION OF SLDC ALGORITHM

APPENDIX A -- DESCRIPTION OF SCAN LINE DIFFERENCE COMPRESSION ALGORITHM

A-1. INTRODUCTION

This Appendix describes a Scan Line Difference Compression (SLDC) algorithm which is hypothesized to be an efficient distortionless data compression algorithm as well as a technique which can easily be implemented using existing microcomputer and LSI technology. An experimental test of this technique is about to be initiated. Until these experimental results are available, it shall be necessary to rely on "arm-waving" arguments to explain why the SLDC algorithm is considered to be a viable candidate for facsimile data compression. These arguments are given in Section A-3.

A-2. DESCRIPTION OF SLDC ALGORITHM

Figure A-1. is a diagram of the overall image transmission system. As shown on this figure, the overall system is partitioned into:

1. the source coding subsystem, concerned with the minimization of the number of bits necessary to permit the distortionless reconstruction of an image, and,
2. the channel coding subsystem, which attempt to correct whatever errors may be introduced during the signal transmission process.

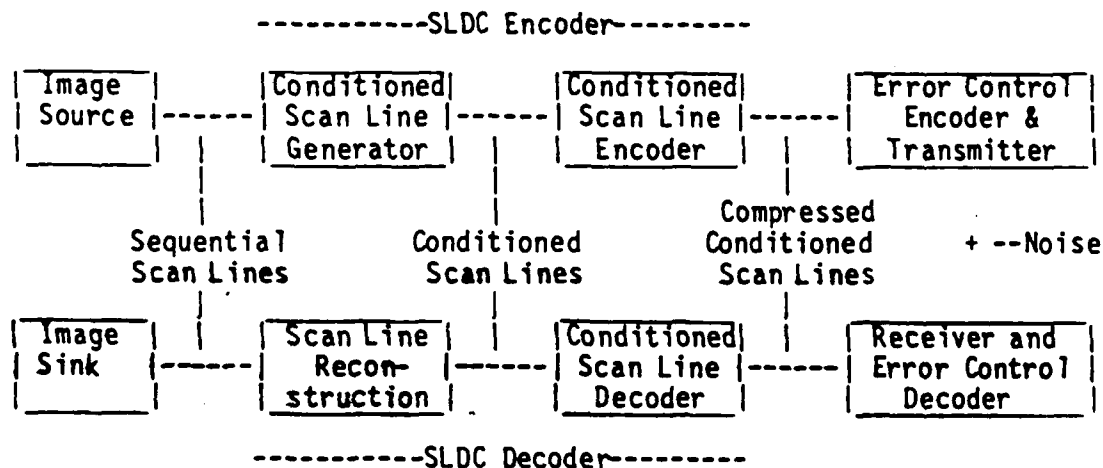


Figure A-1. -- Overall Data Compression Process

This Appendix deals exclusively with the source coding aspects and presumes that the transmission process is noiseless either due to the inherent absence of noise or an effective channel coding subsystem providing the necessary error control.

The SLDC system consists of an encoder, located at the image generation source, and a decoder, located at the point where the image is regenerated. The encoder and decoder represents a transform pair in which the decoder performs the inverse process to that performed by the encoder. For most of the remaining discussion, the SLDC algorithm shall be described in terms of the encoding functions only, since that effectively defines the entire process.

It is assumed that the image to be compressed can be represented as a two dimensional rectangular binary matrix composed of "r" rows and "c" columns. Every matrix element is either white (0) or black (1). The raw image can therefore be expressed with a binary sequence of rc bits in length. The image is output from the source as r successive scan lines, where each scan line consists of a string of c bits.

If the process generating the image were memoryless (i.e., future image elements were independent of all past image elements) and if the probability of a "1" or a "0" at each element were 0.5, then distortionless compression would not be effective and no better scheme than the transmission of the raw rc image bits would be possible. However, in typical images to be transmitted, there is considerable inherent redundancy in the raw image. In these cases, it is generally possible to transmit an encoded replica of the image with T bits (T less than rc) such that the image may be exactly reconstructed using only the T transmitted bits. The efficiency of the compression process can be measured by the compression ratio, R_C , given by

$$R_C = rc/T \quad [EQ. A-1]$$

The SLDC algorithm for the distortionless compression of an image consists of a Conditioned Scan Line Computation step and a Compression step.

A-2.1 Conditioned Scan Line Computation Step.

The purpose of this step is to process an image consisting of a number of scan lines and to generate a Conditioned Image File (CIF), composed of a set of Conditioned Scan Lines (CSLs), subject to the following conditions:

1. the original image can be reconstructed from the CIF without distortion, and,
2. the information theoretic entropy of each CSL is approximately minimized subject to practical processing limitations.

Although these two conditions may appear to be contradictory, they really are not. The first condition requires that the CIF, consisting of the set of CSLs, must be transformable back to the set of original scan lines without distortion. The second condition is concerned only with the compressability of the image considered one scan line at a time. The conditioning procedure should transform the original scan line into a CSL having lower entropy than the original when considered on an individual scan line basis.

The conditioning algorithm shall generate a prediction of a scan line element based upon the state of the four neighboring and previously scanned image elements as shown in Figure A-2.

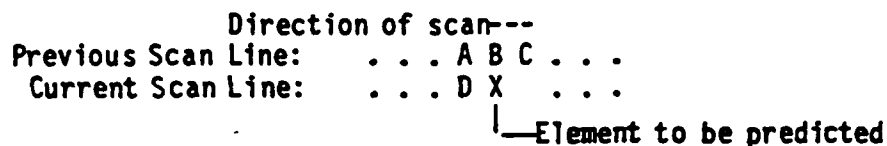


Figure A-2 -- Nearest Neighbor Elements

The CIFs shall be generated based upon a State Machine specification that shall predict the state of each scan element based upon Bits A, B, C, and D where "0" and "1" represent the "expected" and "unexpected" state respectively. A ring of "0" bits shall be presumed to encircle the perimeter of the raw image. This is relevant only when the element being predicted is on the image boundary. The prediction condition algorithm shall be completely defined by a state table that has not yet been determined.

A-2.2 Compression Encoding Step.

The second step of the SLDC encoding process represents the actual compression process. For each Conditioned Scan Line (CSL) string to be compressed, the following substeps shall occur:

1. Encode the weight (W) of CSL. If W=0, proceed to next CSL; otherwise, proceed to Substep 2.) below.
2. Segment CSL into m segments of n-bits each ($mn=c$) where n is a design parameter [NOTE: if c is prime or not divisible by a convenient integer, it may be necessary to pad CSL by appending some "0"-bits in order to achieve a convenient divisor], and,
3. Compute and encode for transmission a Segment State Word (SSW) for each of the m segments. Each of the SSW shall uniquely define the state of the respective segment and shall, in general, be encoded into fewer than n-bits. A SSW contains the following two variable length fields:
 - a. WEIGHT, the weight of the segment, and,
 - b. RANK, an index value which uniquely specifies the bit positions of all "1"-bits within segment. If the WEIGHT-value = 0, the RANK-field is omitted.

After the last non-null segment of a CSL has been encoded, the remaining SSWs may be omitted. This can be recognized by maintaining a running count of processed "1"-bits within the CSL and comparing this to W. The format of an Encoded Scan Line (ESL) is shown in Figure A-3.

Scan Line Weight	SSW(1)	SSW(2)	. . .	SSW(J)	. . .	SSW(n)
------------------------	--------	--------	-------	--------	-------	--------

WEIGHT(J)	RANK(J)
-----------	---------

Figure A-3. -- Encoded Scan Line Format

The key elements of the SLDC encoding consists of the efficient manner in which the WEIGHT(.) and RANK(.) are encoded. This is described in the following two sections.

A-2.2.1 Segment Weight Encoding Procedure.

Let p be the average probability of a "1"-bit within a CSL. By definition,

$$p = W/mn \quad [\text{EQ. A-2}]$$

The segment weight shall be encoded using a variable length Huffman code. This type of code is extensively described in most information theory texts (e.g., Ref. 2). If P_k is defined to be the probability that the weight of an n -bit segment is k , then Huffman coding shall minimize the average length of the segment weight field. Thus,

$$\sum_{k=0}^n P_k L_k = \text{minimum} \quad [\text{EQ. A-3}]$$

where L_k = number of bits in WEIGHT-field specifying a segment of weight k .

For reasons to be explained in Section A.2.2.2, the maximum weight segment that shall be handled by the normal SLDC process is k' where k' is listed in Table A-1. Segments of weight greater than k' shall be handled by special procedures to be described later.

It shall be assumed that the W "1"-bits are binomially distributed within CSL. Hence,

$$P_k = \text{Prob}[\text{segment weight} = k] = C(n, k) p^k (1-p)^{n-k} \quad [\text{EQ. A-4}]$$

$$\text{where } C(n, k) = \frac{n!}{k!(n-k)!}$$

The binary Huffman code is computed for the set of events with respective probabilities,

$$P_0, P_1, \dots, P_{k'}, \text{ and } S_{k'}$$

$$\text{where } S_{k'} = 1 - \sum_{k=0}^{k'} P_k. \quad [\text{EQ. A-5}]$$

After the codebook has been computed, the codewords shall be used to define the segment lengths for the current CSL. Note that p shall typically vary from one scan line to the next and therefore either a set of codebooks must be prestored corresponding to all possible values of p or the codebook must be recomputed each scan line.

For segments whose weight (k) is less than or equal to k' , the segment weight is identified by use of the codeword corresponding to probability $P_{k'}$. If the segment weight exceeds k' , then the codeword for probability $S_{k'}$ shall be encoded followed by the n -bit segment in uncompressed form. In this case the rank encoding process, described in the following section is omitted.

A-2.2.2 Rank Encoding Procedure.

The rank of an n -bit segment of weight k shall be defined to be the number of possible n -bit sequences of weight k that are numerically less than the segment being encoded. The rank encoding process can be implemented recursively by successively scanning the bits of the segment from most significant bit to least significant bit and, whenever a "1"-bit is encountered, to accumulate a count of the number of patterns which are already known to be numerically inferior to that of the segment. The rank field is omitted if $k=0$ and, if k is greater than 0, the rank is encoded into a field of length $[\log_2 C(n,k)]$ -bits where $[X]$ represents the smallest integer greater than or equal to X .

The following example may help explain the procedure. Let the segment length and weight be n and k respectively (k greater than 0). The process is initialized by clearing a counter, named RANK, to zero. Then, starting at the most significant bit, the process recursively scans the segment from left to right as shown in Figure A-4, and increments RANK by an appropriate constant whenever a "1"-bit is encountered. For example, suppose the scanning process has progressed to the point where Bit- I is about to be examined and that j "1"-bits (0 to $j-1$) have previously been encountered in the scanning process from Bit- $(n-1)$ to Bit- $(I+1)$ inclusively. There are therefore $(k-j)$ "1"-bits within the segment that have not yet been discovered. Bit- I is now examined and, if Bit- $I = 0$, the scan process immediately moves to the next bit, Bit- $(I-1)$. If Bit- $I = 1$, then the current segment is known to be numerically superior to all $C(I, k-j)$ sequences which have their low order $(k-j)$ "1"-bits confined to the I low order bit positions (i.e., Bit-0 to Bit $I-1$ inclusive). Therefore RANK shall be incremented by $C(I, k-j)$ and j shall be incremented by 1 (in that order) before proceeding to the next scan position.

The rank encoding process terminates when either all k "1"-bits have been found by the scanning procedure OR there are exactly j "1"-bits that have not yet been found AND there are only j bit positions remaining to be scanned. In the latter case, all unscanned bit positions must contain "1" and therefore have the lowest possible rank. Hence, since the rank increment would be zero, the remaining scanning steps can be bypassed. Annex-1 contains a numerical example of the rank encoding process.

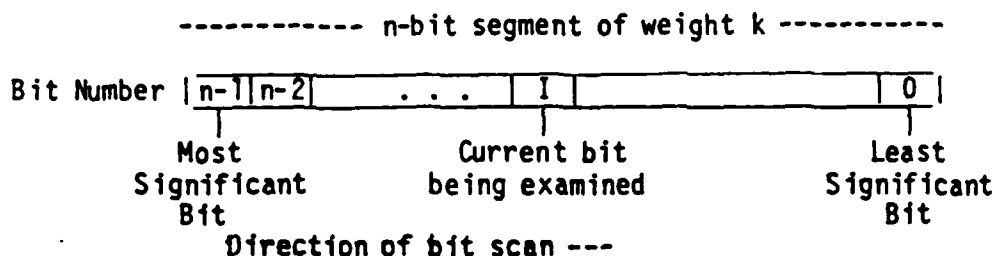


Figure A-4. -- Rank Encoding Scanning Process

It shall be presumed that the encoding algorithm is capable of performing arithmetic integer computation with a maximum word length of either 16 or 32 bits. If the segment length is greater than the arithmetic word length, the RANK-value might cause an arithmetic overflow. For this reason, the value of k' shall be limited to the following maximum values to avoid the possibility of an overflow condition.

Segment length in bits	Maximum value of k'	
	16-bit word length unconstrained	32-bit wordlength unconstrained
18 or less		
20	6	"
24	5	"
28	4	"
32	4	"
36	4	14
40	3	11
44	3	10
48	3	9
52	3	8
56	3	8
60	3	8
64	3	7

Table A-1. - Maximum Weight Segment for Normal SLDC Processing

A-3 RATIONALE FOR SLDC ALGORITHM

The effectiveness of a facsimile data compression system must necessarily be based upon the performance/cost tradeoff with respect to documents representing a realistic traffic load. It is the purpose of the present investigation to subject the SLDC algorithm to such a test using document images believed to be representative of commercial requirements. Until the results of this investigation are completed, the effectiveness of the SLDC algorithm remains in doubt.

Since the forthcoming investigation shall require money and other resources, it is reasonable to insist upon some rationale for the potential benefits that may be gained before embarking on this investigation. There are an infinite number of possible compression algorithms. Why should the Government expend resources to investigate the SLDC technique? While an insistence on a "proof-of-effectiveness" as a prerequisite for the investigation is impossible and would represent a Catch-22 dilemma, the following is offered as a plausibility argument in defense of the SLDC algorithm.

The SLDC represents a two step process. In the first step the image is reduced to a set of conditioned scan lines (CSLs), each of which are constructed so that they contain only the "unexpected" changes in image pattern relative to the previously scanned neighboring image elements. A pattern extension that can be implied from the previously scanned image need not be transmitted since the decoder could reconstruct the image extension without guidance from the encoder. It is only when the decoder would incorrectly construct the image that guidance is needed from the encoder to override the decoder's default algorithm. Unexpected changes are signalled by a binary "1"; the absence of an unexpected change is encoded as a "0". It is expected that the investigation performed under the Section 3.1 subtask shall develop an efficient procedure that minimizes the frequency of unexpected changes. It seems reasonable that a conditioning technique which predicts the binary state of an image element based upon some weighted average of four of the element's nearest neighbors should be correct most of the time.

The second step of the SLDC algorithm represents the actual compression process. Each CSL to be compressed is fragmented into a series of n -bit segments. For each segment a variable length Huffman code is used to encode the weight (k) of the segment. Huffman codes are known to be optimal in the sense that no other distortionless code is possible which can encode the data into a lesser average number of bits. Assuming that each of the possible $C(n,k)$ weight- k patterns of length n -bits are equally likely, the rank encoding procedures is also theoretically optimal in encoding the position of the k "1"-bits in the segment. Thus, each of the components of the encoding process is optimal in some sense; however, this does not imply that the entire algorithm is globally optimal. Indeed it is intuitively obvious that some non-optimality is introduced at each stage of modularization. The SLDC algorithm has performed the following modularization steps:

1. the fragmentation of an image into CSLs,
2. the fragmentation of a CSL into segments, and,
3. the separate encoding of segment weight and specific segment pattern given a known weight.

While each of these partitioning operations shall undoubtedly reduce the compression ratio performance relative to a globally optimal compressor, the modularization is expected to reduce the computational complexity and may therefore allow the SLDC algorithm to approach the globally optimal compression scheme more closely than previously implemented compression scheme. Hence, although the performance and feasibility of implementation cannot be proven at this point, the SLDC procedure offers considerable promise and is considered worthy of a empirical demonstration which shall evaluate its effectiveness relative to other compression schemes.

ANNEX-1 -- NUMERICAL EXAMPLE OF RANK ENCODING PROCESS

In this Annex the rank encoding of a 9-bit segment of weight 3 shall be evaluated. In this case, $n=9$ and $k=3$. Suppose the segment pattern were 010100100 and let us establish the bit numbering convention as shown below.

Bit Number:	8	7	6	5	4	3	2	1	0
Bit State:	0	1	0	1	0	0	1	0	0

Initialize by setting RANK=0, $j=0$ (Number of previously encountered "1" bits), and the bit scan position to Bit-8.

Scan Position (I)	Bit State	No. of Found "1"-Bits (j)	$C(I, k-j)$	Rank Encoding computation
		0		RANK = 0 (INITIALIZATION)
8	0	0	56	RANK = 0
7	1	0	35	RANK = 0 + 35 = 35
6	0	1	15	RANK = 35
5	1	1	10	RANK = 35 + 10 = 45
4	0	2	4	RANK = 45
3	0	2	3	RANK = 45
2	1	2	2	RANK = 45 + 2 = 47
1	0	3 ($j = k$: TERMINATE with RANK = 47)		

The encoded length of the RANK field is $\lceil \log_2 C(9,3) \rceil = \lceil \log_2 84 \rceil = \lceil 6.394 \rceil = 7$. Thus the RANK-field is encoded as 0101111.

To validate this example, the table below lists all possible $n=9$, $k=3$ patterns in numerically increasing order up to and including the test case (010100100).

RANK	PATTERN	RANK	PATTERN	RANK	PATTERN
0	000000111	16	000110001	32	001100100
1	000001011	17	000110010	33	001101000
2	000001101	18	000110100	34	001110000
3	000001110	19	000111000	35	010000011
4	000010011	20	001000011	36	010000101
5	000010101	21	001000101	37	010000110
6	000010110	22	001000110	38	010001001
7	000011001	23	001001001	39	010001010
8	000011010	24	001001010	40	010001100
9	000011100	25	001001100	41	010010001
10	000100011	26	001010001	42	010010010
11	000100101	27	001010010	43	010010100
12	000100110	28	001010100	44	010011000
13	000101001	29	001011000	45	010100001
14	000101010	30	001100001	46	010100010
15	000101100	31	001100010	47	010100100 -- Test Case Segment

APPENDIX B

SOFTWARE OPERATING INSTRUCTIONS

PROGRAM: CHART

DESCRIPTION:

This program reads an image input file one line at a time recording occurrences of each nearest neighbor state vector and state of each bit for every bit in the file. The results are sent to the line printer in the form of Table 1 in Appendix A. Also an occurrence-black-white table of the same nature is calculated.

CALLING SEQUENCE

CHART, <INPUT NAME>, <PROBABILITY FILE NAME>

INPUT NAME - Input Image File

OUTPUT NAME - File consisting of a "1" or "0" for each neighbor template depending on probabilities calculated. This file will be used in Conditioned Image File creation and restoration.

ORDER OF PARAMETERS:

- 1) Number of records in input file
- 2) CCITT File #
- 3) File Resolution

PROGRAM: CREATE

DESCRIPTION:

This program creates a CIF along with statistics requested in subtask 1 (see Section 2.0). The file generated in CHART is read in and used to generate the CIF.

CALLING SEQUENCE

CREATE, <INPUT NAME>, <OUTPUT NAME>, <PROBABILITY FILE>

INPUT NAME - Input Name of Image

OUTPUT NAME - Output Name of CIF

PROBABILITY FILE - File Generated by CHART

ORDER OF PARAMETERS

- 1) # Word per output record
- 2) # records to be output
- 3) CCITT file number
- 4) File resolution
- 5) # records in input file

PROGRAM: RESTORE

DESCRIPTION:

This program restores the CIF back to the original image file. The probability file used to generate the CIF is read in and used to restore the file to its original state.

CALLING SEQUENCE:

RESTORE - <INPUT NAME>, <OUTPUT NAME>, <PROBABILITY FILE>

INPUT NAME - Input CIF Name

OUTPUT NAME - Restored Output Image File

PROBABILITY FILE - File used to create CIF; file is calculated
in program CHART.

ORDER OF PARAMETERS

- 1) # of words to be output
- 2) # records to be output
- 3) # records in input file

PROGRAM: ENCODE

DESCRIPTION:

This program encodes an input CIF into an output encoded file. Before encoding each line, a codebook, as discussed in Section 3.0, is generated and stored. The output file will be written to disk if an output file is requested. There is also an option to print line by line compression statistics.

CALLING SEQUENCE

ENCODE - <INPUT NAME>, <OUTPUT NAME>, <CODE FILE>,
 <WEIGHT FILE>

INPUT NAME - Input CIF Name

OUTPUT NAME - Output Encoded File Name

ORDER OF PARAMETERS

- 1) # records per output file
- 2) Decision for output file
- 3) Max weight for segment (Table 3.1)
- 4) CCITT file number
- 5) File resolution
- 6) Arithmetic word length
- 7) # words per input record
- 8) # records in input file
- 9) Segment length to be compressed
- 10) Decision for line by line compression stats.

PROGRAM: DECODE

DESCRIPTION:

Program DECODE decodes the encoded file back into a CIF. The prestored codebook generated in program encode is read in and used for decoding the segment weights.

A linked Huffman decoding tree is read in which finds the Huffman code used for the segment weight coding. The Huffman code is then looked up in the prestored codebook to find the segment weight. Once the weight of the segment is found the rank can be easily calculated.

CALLING SEQUENCE:

DECODE <INPUT NAME>, <OUTPUT NAME>, <WEIGHT FILES>, <TREE FILES>

INPUT NAME - Encoded Input CIF Name

OUTPUT NAME - Decoded Output CIF name

ORDER OF PARAMETERS

- 1) # words per output record
- 2) # records to be output
- 3) # words per input record
- 4) # records in input file
- 5) Segment length to be compressed
- 6) Max weight encodable (Table 3.1)

APPENDIX C

SOFTWARE CODE LISTINGS

```

3 SFILS 3,4,5
4
5 PROGRAM CHART
6
7 C*****
8 C
9 C AUTHOR: J. P. DIMAGGIO
10 C
11 C DESCRIPTION:
12 C
13 C This program develops a table of the number of occurrences of
14 C neighboring bit patterns and the state of the current bit.
15 C It also develops a probability table and output file with
16 C respective probabilities of bit patterns and bit states.
17 C
18 C
19 C CALLING SEQUENCE:
20 C
21 C RU,CHART,<INPUT NAMR>,<OUTPUT NAMR>
22 C
23 C*****
24 C
25 C INPUT PARAMETERS:
26 C
27 C NUMBER OF RECORDS TO PROCESS
28 C CCITT FILE NUMBER
29 C FILE RESOLUTION
30 C
31 C
32 C
33 C IMPLICIT NONE
34 C
35 C
36 C
37 C DEFINE PARAMETERS
38 C
39 C INTEGER MAXLIN I MAXIMUM NO. OF OUTPUT RECORDS
40 C INTEGER BUFD I MAXIMUM NO. OF WORDS PER INPUT RECORD
41 C INTEGER BITS I MAX NO. OF BITS PER RECORD
42 C INTEGER DERECS I HOW MANY RECORDS TO BE PROCESSED
43 C
44 C
45 C PARAMETER (BUFD=512)
46 C PARAMETER (BITS=3500)
47 C
48 C ***** FILE DEFINITIONS *****
49 C
50 C INTEGER ISTAT I I-O STATUS VARIABLE
51 C INTEGER LBUF(BUFD) I READ/WRITE BUFFER FOR FORTRAN 77
52 C INTEGER BUFR(BUFD) I RECORD BUFFER
53 C INTEGER*4 MREC I NO. OF RECORDS IN INPUT FILE
54 C INTEGER*4 BUFRMAX I INPUT RECORD SIZE IN WORDS
55 C
56 C ***** LOCAL VARIABLES *****
57 C
58 C INTEGER PREV(-2,BITS) I HOLDS BIT VALUES OF PREVIOUS LINE READ

```



```

58 INTEGER TABLE(16,4)      I TABLE OF NEIGHBORING BIT PATTERNS
59 REAL PROB(16,2)           I PROBABILITY ARRAY
60 INTEGER*4 COUNT(16,3)     I RESULT OCCURRENCE ARRAY
61 INTEGER*4 TOTTOT          I TOTAL BIT COUNT
62 INTEGER*4 TOTBLK          I TOTAL BLACK BIT COUNT
63 INTEGER*4 TOTWHT          I TOTAL WHITE BIT COUNT
64 INTEGER NBR(4)            I HOLDS VALUES OF NEIGHBORING BITS
65 INTEGER BITVAL            I VALUE OF CURRENT BIT
66 INTEGER PREBIT           I VALUE OF PREVIOUS BIT
67 REAL BITNUM              I NUMBER OF BITS IN FILE
68 INTEGER BITWRD            I PRESENT WORD IN BUFFER
69 INTEGER CT               I INDEX OF CURRENT BIT IN LINE
70 P.I.J.K.L.O             I LOOP COUNTERS
71 INTEGER FLNUM             I CCITT FILE NUMBER
72 INTEGER RESLTN            I CURRENT FILE RESOLUTION
73 INTEGER FIND              I INDEX USED IN SEARCH
74 INTEGER MUL               I NARROWING VARIABLE USED IN SEARCH
75 C
76 C***** FILE PARAMETERS *****
77 C
78 CHARACTER INFILE*2#
79 CHARACTER ACCTYP*2#
80 CHARACTER DTFIL*2#
81 INTEGER IRCL,OREC
82 INTEGER ITBUF(15)
83 INTEGER TERM
84 INTEGER LFIL
85 INTEGER OUTFIL
86 INTEGER PELFIL
87 INTEGER ITLOG
88 LOGICAL EXISTS
89
90 DATA TERM,LFIL,PELFIL,OUTFIL/1,6,7,8/

```

```

I FILE PARAMETERS
I ARRAY FOR TIME-OF-DAY
I TERMINAL LU
I PRINTER LU
I OUTPUT FILE LU
I LU OF INPUT FILE
I LIBRARY FUNCTION

```

```

92 C ***** INITIALIZING *****
93 C
94 C
95 DO I=-2,BITS
96   PREV(I)=0
97 END DO
98
99 DO I=1,16
100   DO J=1,3
101     COUNT(I,J)=0
102   END DO
103 END DO
104 C
105 C ***** BEGIN PROGRAM *****
106 C
107 C GET INPUT FILE NAME
108 C
109 CALL RCPAR(1,INFILE)
110 INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
111 & EXIST=EXIST, ACCESS=ACCTYP, IOSTAT=ISTAT)
112 IF(ISTAT.NE.0) STOP 'INQUIRE ERROR'
113 IF(EXIST) THEN
114   IF(ACCTYP.EQ.'SEQUENTIAL') THEN
115     IF(IRCL.EQ.0.AND.MREC.EQ.0) THEN
116       OPEN(UNIT=PELFIL,FILE=INFILE,ACCESS=ACCTYP)
117       CALL LGBUF(LBUF,BUFD)
118       READ(UNIT=PELFIL,IOSTAT=ISTAT) (BUFFER(I),I=1,BUFD*1)
119       IF(ISTAT.NE.0) THEN
120         IRCL=ITFLOG(1)
121         MREC=MAXLIN
122         BACKSPACE(UNIT=PELFIL,IOSTAT=ISTAT)
123         IF(ISTAT.NE.0) STOP 'BACKSPACE ERROR'
124       ELSE
125         STOP 'NO ERROR IS AN ERROR'
126       END IF
127     ELSE
128       STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
129     END IF
130   ELSE IF(ACCTYP.EQ.'DIRECT') THEN
131     OPEN(UNIT=PELFIL,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT',
132 & RECL=IRCL,MAXREC=MREC)
133   ELSE
134     STOP 'ACCTYP UNDEFINED'
135   END IF
136 ELSE
137   STOP 'INPUT FILE DOES NOT EXIST'
138 END IF
139 C ***** READ IN INPUT PARAMETERS *****
140 C
141 C
142 WRITE(TERM,*)('HOW MANY RECORDS TO PROCESS?')
143 READ(TERM,*) DERECS
144 MREC=MIN(MREC,DERECS)
145 WRITE(TERM,*) 'CCITT IMAGE NUMBER?'
146 READ(TERM,*) FLNUM

```

```

147 WRITE(TERM,*) 'FILE RESOLUTION?'
148 READ(TERM,*) RESLTN
149 C***** BEGIN FILE PROCESSING *****
150 C
151 C
152 BUFMAX=IRCL/2
153
154 DO I=1,MREC
155
156 CT=1
157 PREBIT=0
158
159 CALL LGBUF(LBUF,BUFD)
160 READ(UNIT=PELFI,IOSTAT=ISTAT) (BUFFER(P),P=1,BUFMAX)
161 IF(ISTAT.NE.0) THEN
162 WRITE(LFIL,('INPUT READ ERROR. ISTAT=',I4)) ISTAT
163 STOP
164 END IF
165
166 DO K=1,BUFMAX
167
168 BITWRD=BUFFER(K)
169
170 DO L=1,8,-1
171
172 IF (BTEST(BITWRD,L))THEN
173 BITVAL=1
174 ELSE
175 BITVAL=0
176 END IF
177 DO Q=1,3
178 NBR(Q)=PREV(Q-1)+(CT-1)
179 END DO
180 NBR(4)=PREBIT
181 PREV(CT-1)=PREBIT
182 PREBIT=BITVAL
183 CT=CT+1
184 C***** INCREMENT PROPER NEIGHBOR TEMPLATE *****
185 C
186 C
187 FIND=1
188 MUL=16
189 DO P=1,4
190 IF (NBR(P).EQ.1) THEN
191 FIND=FIND*MUL/2
192 END IF
193 MUL=MUL/2
194 END DO
195
196 COUNT(FIND,1)=COUNT(FIND,1)+1
197 IF (BITVAL.EQ.1) THEN
198 COUNT(FIND,2)=COUNT(FIND,2)+1
199 ELSE
200 COUNT(FIND,3)=COUNT(FIND,3)+1
201 END IF

```

```

282          END DO          I NUMBER BITS IN WORD
283
284          END DO          I NUMBER OF WORDS PER RECORD
285
286          PREVICT-1)-BITVAL
287
288          END DO          I NUMBER OF RECORDS PER LINE
289
290          ***** SET UP TABLE *****
291 C
292 C
293 C
294 C
295 C
296 C
297 C
298 C
299 C
300 C
301 C
302 C
303 C
304 C
305 C
306 C
307 C
308 C
309 C
310 C
311 C
312 C
313 C
314 C
315 C
316 C
317 C
318 C
319 C
320 C
321 C
322 C
323 C
324 C
325 C
326 C
327 C
328 C
329 C
330 C
331 C
332 C
333 C
334 C
335 C
336 C
337 C
338 C
339 C
340 C
341 C
342 C
343 C
344 C
345 C
346 C
347 C
348 C
349 C
350 C
351 C
352 C
353 C
354 C
355 C
356 C
357 C
358 C
359 C
360 C
361 C
362 C
363 C
364 C
365 C
366 C
367 C
368 C
369 C
370 C
371 C
372 C
373 C
374 C
375 C
376 C
377 C
378 C
379 C
380 C
381 C
382 C
383 C
384 C
385 C
386 C
387 C
388 C
389 C
390 C
391 C
392 C
393 C
394 C
395 C
396 C
397 C
398 C
399 C
400 C
401 C
402 C
403 C
404 C
405 C
406 C
407 C
408 C
409 C
410 C
411 C
412 C
413 C
414 C
415 C
416 C
417 C
418 C
419 C
420 C
421 C
422 C
423 C
424 C
425 C
426 C
427 C
428 C
429 C
430 C
431 C
432 C
433 C
434 C
435 C
436 C
437 C
438 C
439 C
440 C
441 C
442 C
443 C
444 C
445 C
446 C
447 C
448 C
449 C
450 C
451 C
452 C
453 C
454 C
455 C
456 C
457 C
458 C
459 C
460 C
461 C
462 C
463 C
464 C
465 C
466 C
467 C
468 C
469 C
470 C
471 C
472 C
473 C
474 C
475 C
476 C
477 C
478 C
479 C
480 C
481 C
482 C
483 C
484 C
485 C
486 C
487 C
488 C
489 C
490 C
491 C
492 C
493 C
494 C
495 C
496 C
497 C
498 C
499 C
500 C
501 C
502 C
503 C
504 C
505 C
506 C
507 C
508 C
509 C
510 C
511 C
512 C
513 C
514 C
515 C
516 C
517 C
518 C
519 C
520 C
521 C
522 C
523 C
524 C
525 C
526 C
527 C
528 C
529 C
530 C
531 C
532 C
533 C
534 C
535 C
536 C
537 C
538 C
539 C
540 C
541 C
542 C
543 C
544 C
545 C
546 C
547 C
548 C
549 C
550 C
551 C
552 C
553 C
554 C
555 C
556 C
557 C
558 C
559 C
560 C
561 C
562 C
563 C
564 C
565 C
566 C
567 C
568 C
569 C
570 C
571 C
572 C
573 C
574 C
575 C
576 C
577 C
578 C
579 C
580 C
581 C
582 C
583 C
584 C
585 C
586 C
587 C
588 C
589 C
590 C
591 C
592 C
593 C
594 C
595 C
596 C
597 C
598 C
599 C
600 C
601 C
602 C
603 C
604 C
605 C
606 C
607 C
608 C
609 C
610 C
611 C
612 C
613 C
614 C
615 C
616 C
617 C
618 C
619 C
620 C
621 C
622 C
623 C
624 C
625 C
626 C
627 C
628 C
629 C
630 C
631 C
632 C
633 C
634 C
635 C
636 C
637 C
638 C
639 C
640 C
641 C
642 C
643 C
644 C
645 C
646 C
647 C
648 C
649 C
650 C
651 C
652 C
653 C
654 C
655 C
656 C
657 C
658 C
659 C
660 C
661 C
662 C
663 C
664 C
665 C
666 C
667 C
668 C
669 C
670 C
671 C
672 C
673 C
674 C
675 C
676 C
677 C
678 C
679 C
680 C
681 C
682 C
683 C
684 C
685 C
686 C
687 C
688 C
689 C
690 C
691 C
692 C
693 C
694 C
695 C
696 C
697 C
698 C
699 C
700 C
701 C
702 C
703 C
704 C
705 C
706 C
707 C
708 C
709 C
710 C
711 C
712 C
713 C
714 C
715 C
716 C
717 C
718 C
719 C
720 C
721 C
722 C
723 C
724 C
725 C
726 C
727 C
728 C
729 C
730 C
731 C
732 C
733 C
734 C
735 C
736 C
737 C
738 C
739 C
740 C
741 C
742 C
743 C
744 C
745 C
746 C
747 C
748 C
749 C
750 C
751 C
752 C
753 C
754 C
755 C
756 C
757 C
758 C
759 C
760 C
761 C
762 C
763 C
764 C
765 C
766 C
767 C
768 C
769 C
770 C
771 C
772 C
773 C
774 C
775 C
776 C
777 C
778 C
779 C
780 C
781 C
782 C
783 C
784 C
785 C
786 C
787 C
788 C
789 C
790 C
791 C
792 C
793 C
794 C
795 C
796 C
797 C
798 C
799 C
800 C
801 C
802 C
803 C
804 C
805 C
806 C
807 C
808 C
809 C
810 C
811 C
812 C
813 C
814 C
815 C
816 C
817 C
818 C
819 C
820 C
821 C
822 C
823 C
824 C
825 C
826 C
827 C
828 C
829 C
830 C
831 C
832 C
833 C
834 C
835 C
836 C
837 C
838 C
839 C
840 C
841 C
842 C
843 C
844 C
845 C
846 C
847 C
848 C
849 C
850 C
851 C
852 C
853 C
854 C
855 C
856 C
857 C
858 C
859 C
860 C
861 C
862 C
863 C
864 C
865 C
866 C
867 C
868 C
869 C
870 C
871 C
872 C
873 C
874 C
875 C
876 C
877 C
878 C
879 C
880 C
881 C
882 C
883 C
884 C
885 C
886 C
887 C
888 C
889 C
890 C
891 C
892 C
893 C
894 C
895 C
896 C
897 C
898 C
899 C
900 C
901 C
902 C
903 C
904 C
905 C
906 C
907 C
908 C
909 C
910 C
911 C
912 C
913 C
914 C
915 C
916 C
917 C
918 C
919 C
920 C
921 C
922 C
923 C
924 C
925 C
926 C
927 C
928 C
929 C
930 C
931 C
932 C
933 C
934 C
935 C
936 C
937 C
938 C
939 C
940 C
941 C
942 C
943 C
944 C
945 C
946 C
947 C
948 C
949 C
950 C
951 C
952 C
953 C
954 C
955 C
956 C
957 C
958 C
959 C
960 C
961 C
962 C
963 C
964 C
965 C
966 C
967 C
968 C
969 C
970 C
971 C
972 C
973 C
974 C
975 C
976 C
977 C
978 C
979 C
980 C
981 C
982 C
983 C
984 C
985 C
986 C
987 C
988 C
989 C
990 C
991 C
992 C
993 C
994 C
995 C
996 C
997 C
998 C
999 C
1000 C

```

```

257 TOTBLK=TOTBLK+COUNT(I,2)
258 TOTWHT=TOTWHT+COUNT(I,3)
259 IF (COUNT(I,1).EQ.0) THEN
260   PROB(I,2)=0.0
261 ELSE
262   PROB(I,2)=REAL(COUNT(I,3))/REAL(COUNT(I,1))
263 END IF
264 COUNT(I,1)=REAL(COUNT(I,1))
265 PROB(I,1)=COUNT(I,1)/BITNUM
266 END DO
267
268 100 FORMAT (4(I2),IX,3(I10))
269 WRITE(LFIL,125) TOTTOT, TOTBLK, TOTWHT
270 125 FORMAT (1X, 'TOTAL', 11(I10,110))
271
272 C***** PRINT HEADINGS *****
273 C*****
274 C
275 WRITE(LFIL,*)(' STATE')
276 WRITE(LFIL,*)('A B C D PROBABILITY PROB(X=STATE)')
277 WRITE(LFIL,*)('*****')
278 C***** PRINT PROBABILITY RESULTS *****
279 C*****
280 C
281 200 WRITE(LFIL,200)((TABLE(I,J),J=1,4),(PROB(I,J),J=1,2),I=1,16)
282 200 FORMAT (16(4(I2),(F14.4),(F17.4)/))
283 CALL RCPAR(2,DTFIL)
284 OPEN(UNIT=OUTFIL,FILE=DTFIL)
285 DO I=1,16
286   IF (PROB(I,2).GE.0.5) THEN
287     BITVAL=0
288     WRITE(OUTFIL,300) (BITVAL)
289   ELSE
290     BITVAL=1
291     WRITE(OUTFIL,300) (BITVAL)
292   END IF
293 END DO
294
295 300 FORMAT(I3)
296
297 C*****
298 C
299 C
300 C
301 C
302 C
303 C
304 C
305 C
306 C
307 C
308 C
309 C
310 C
311 C
312 C
313 C
314 C
315 C
316 C
317 C
318 C
319 C
320 C
321 C
322 C
323 C
324 C
325 C
326 C
327 C
328 C
329 C
330 C
331 C
332 C
333 C
334 C
335 C
336 C
337 C
338 C
339 C
340 C
341 C
342 C
343 C
344 C
345 C
346 C
347 C
348 C
349 C
350 C
351 C
352 C
353 C
354 C
355 C
356 C
357 C
358 C
359 C
360 C
361 C
362 C
363 C
364 C
365 C
366 C
367 C
368 C
369 C
370 C
371 C
372 C
373 C
374 C
375 C
376 C
377 C
378 C
379 C
380 C
381 C
382 C
383 C
384 C
385 C
386 C
387 C
388 C
389 C
390 C
391 C
392 C
393 C
394 C
395 C
396 C
397 C
398 C
399 C
400 C
401 C
402 C
403 C
404 C
405 C
406 C
407 C
408 C
409 C
410 C
411 C
412 C
413 C
414 C
415 C
416 C
417 C
418 C
419 C
420 C
421 C
422 C
423 C
424 C
425 C
426 C
427 C
428 C
429 C
430 C
431 C
432 C
433 C
434 C
435 C
436 C
437 C
438 C
439 C
440 C
441 C
442 C
443 C
444 C
445 C
446 C
447 C
448 C
449 C
450 C
451 C
452 C
453 C
454 C
455 C
456 C
457 C
458 C
459 C
460 C
461 C
462 C
463 C
464 C
465 C
466 C
467 C
468 C
469 C
470 C
471 C
472 C
473 C
474 C
475 C
476 C
477 C
478 C
479 C
480 C
481 C
482 C
483 C
484 C
485 C
486 C
487 C
488 C
489 C
490 C
491 C
492 C
493 C
494 C
495 C
496 C
497 C
498 C
499 C
500 C
501 C
502 C
503 C
504 C
505 C
506 C
507 C
508 C
509 C
510 C
511 C
512 C
513 C
514 C
515 C
516 C
517 C
518 C
519 C
520 C
521 C
522 C
523 C
524 C
525 C
526 C
527 C
528 C
529 C
530 C
531 C
532 C
533 C
534 C
535 C
536 C
537 C
538 C
539 C
540 C
541 C
542 C
543 C
544 C
545 C
546 C
547 C
548 C
549 C
550 C
551 C
552 C
553 C
554 C
555 C
556 C
557 C
558 C
559 C
560 C
561 C
562 C
563 C
564 C
565 C
566 C
567 C
568 C
569 C
570 C
571 C
572 C
573 C
574 C
575 C
576 C
577 C
578 C
579 C
580 C
581 C
582 C
583 C
584 C
585 C
586 C
587 C
588 C
589 C
590 C
591 C
592 C
593 C
594 C
595 C
596 C
597 C
598 C
599 C
600 C
601 C
602 C
603 C
604 C
605 C
606 C
607 C
608 C
609 C
610 C
611 C
612 C
613 C
614 C
615 C
616 C
617 C
618 C
619 C
620 C
621 C
622 C
623 C
624 C
625 C
626 C
627 C
628 C
629 C
630 C
631 C
632 C
633 C
634 C
635 C
636 C
637 C
638 C
639 C
640 C
641 C
642 C
643 C
644 C
645 C
646 C
647 C
648 C
649 C
650 C
651 C
652 C
653 C
654 C
655 C
656 C
657 C
658 C
659 C
660 C
661 C
662 C
663 C
664 C
665 C
666 C
667 C
668 C
669 C
670 C
671 C
672 C
673 C
674 C
675 C
676 C
677 C
678 C
679 C
680 C
681 C
682 C
683 C
684 C
685 C
686 C
687 C
688 C
689 C
690 C
691 C
692 C
693 C
694 C
695 C
696 C
697 C
698 C
699 C
700 C
701 C
702 C
703 C
704 C
705 C
706 C
707 C
708 C
709 C
710 C
711 C
712 C
713 C
714 C
715 C
716 C
717 C
718 C
719 C
720 C
721 C
722 C
723 C
724 C
725 C
726 C
727 C
728 C
729 C
730 C
731 C
732 C
733 C
734 C
735 C
736 C
737 C
738 C
739 C
740 C
741 C
742 C
743 C
744 C
745 C
746 C
747 C
748 C
749 C
750 C
751 C
752 C
753 C
754 C
755 C
756 C
757 C
758 C
759 C
760 C
761 C
762 C
763 C
764 C
765 C
766 C
767 C
768 C
769 C
770 C
771 C
772 C
773 C
774 C
775 C
776 C
777 C
778 C
779 C
780 C
781 C
782 C
783 C
784 C
785 C
786 C
787 C
788 C
789 C
790 C
791 C
792 C
793 C
794 C
795 C
796 C
797 C
798 C
799 C
800 C
801 C
802 C
803 C
804 C
805 C
806 C
807 C
808 C
809 C
810 C
811 C
812 C
813 C
814 C
815 C
816 C
817 C
818 C
819 C
820 C
821 C
822 C
823 C
824 C
825 C
826 C
827 C
828 C
829 C
830 C
831 C
832 C
833 C
834 C
835 C
836 C
837 C
838 C
839 C
840 C
841 C
842 C
843 C
844 C
845 C
846 C
847 C
848 C
849 C
850 C
851 C
852 C
853 C
854 C
855 C
856 C
857 C
858 C
859 C
860 C
861 C
862 C
863 C
864 C
865 C
866 C
867 C
868 C
869 C
870 C
871 C
872 C
873 C
874 C
875 C
876 C
877 C
878 C
879 C
880 C
881 C
882 C
883 C
884 C
885 C
886 C
887 C
888 C
889 C
890 C
891 C
892 C
893 C
894 C
895 C
896 C
897 C
898 C
899 C
900 C
901 C
902 C
903 C
904 C
905 C
906 C
907 C
908 C
909 C
910 C
911 C
912 C
913 C
914 C
915 C
916 C
917 C
918 C
919 C
920 C
921 C
922 C
923 C
924 C
925 C
926 C
927 C
928 C
929 C
930 C
931 C
932 C
933 C
934 C
935 C
936 C
937 C
938 C
939 C
940 C
941 C
942 C
943 C
944 C
945 C
946 C
947 C
948 C
949 C
950 C
951 C
952 C
953 C
954 C
955 C
956 C
957 C
958 C
959 C
960 C
961 C
962 C
963 C
964 C
965 C
966 C
967 C
968 C
969 C
970 C
971 C
972 C
973 C
974 C
975 C
976 C
977 C
978 C
979 C
980 C
981 C
982 C
983 C
984 C
985 C
986 C
987 C
988 C
989 C
990 C
991 C
992 C
993 C
994 C
995 C
996 C
997 C
998 C
999 C
1000 C

```

```

Page 1 FTM.                      Opt: 77/LCYI      Wed Aug 14, 1985 11:49:18 am
CIF Generation Program           /SLDC/SLDC1/CREATE.FTN

3 SFILS 3,4,5
4 PROGRAM CREATE
5
6 C*****
7 C
8 C AUTHOR J. P. DIMAGGIO
9 C
10 C DESCRIPTION:
11 C This program creates a conditioned image file using
12 C the probability data generated by PROGRAM CHART.
13 C It also generates all necessary statistics of the
14 C structure of the CIF.
15 C
16 C CALLING SEQUENCE:
17 C
18 C RU,CREATE,<INPUT NAMR>,<OUTPUT NAMR>,<PARAMETER FILE>,<PROBABILITY FILE
19 C C*****
20 C INPUT PARAMETERS:
21 C NUMBER OF WORDS PER OUTPUT RECORD
22 C NUMBER OF RECORDS PER OUTPUT FILE
23 C NUMBER OF RECORDS IN INPUT FILE
24 C
25 C C*****
26 C
27 C IMPLICIT NONE
28 C
29 C
30 C
31 C
32 C
33 C DEFINE PARAMETERS
34 C
35 C INTEGER MAXLIN      I MAXIMUM NO. OF OUTPUT RECORDS
36 C INTEGER DERECS      I RECORDS IN INPUT FILE
37 C INTEGER BUFD        I MAXIMUM NO. OF WORDS PER INPUT RECORD
38 C INTEGER BITS        I MAXIMUM NO. OF BITS PER RECORD RECORD
39 C
40 C PARAMETER (MAXLIN=5000)
41 C PARAMETER (BITS=3500)
42 C PARAMETER (BUFD=512)
43 C
44 C ***** FILE DEFINITIONS *****
45 C
46 C INTEGER ISTAT      I I-O STATUS VARIABLE
47 C INTEGER LBUF(BUFD) I READ/WRITE BUFFER FOR FORTRAN 77
48 C INTEGER BUFR(BUFD) I RECORD BUFFER
49 C
50 C INTEGER LINMAX      I NO. OF RECORDS TO BE OUTPUT
51 C INTEGER BUFDIM      I NO. OF WORDS PER OUTPUT RECORD
52 C INTEGER*4 MREC      I NO. OF RECORDS IN INPUT FILE
53 C
54 C INTEGER*4 BUFRMAX   I INPUT RECORD SIZE IN WORDS
55 C
56 C ***** LOCAL VARIABLES *****
57 C

```

```

58 INTEGER PREV(-1:BITS)
59 INTEGER WEIGHTS
60 INTEGER=4 RNLGTH(B:BITS)
61 INTEGER=4 THWT(B:32)
62 INTEGER=4 SWT(B:64)
63 INTEGER=4 TOTAL
64 INTEGER SIXT
65 INTEGER THRCT
66 INTEGER PROB(16)
67 INTEGER NBR(4)
68 INTEGER BITVAL
69 INTEGER PREBIT
70 INTEGER=4 BITNUM
71 INTEGER=4 PSOLD
72 INTEGER=4 LNWGT
73 INTEGER=4 FLNUM,RESLTN
74 INTEGER=4 FVGT
75 INTEGER=4 OBUFF(BUFD)
76 INTEGER BITWRD
77 INTEGER OBTRD
78 INTEGER CT
79 INTEGER P I,K,L,Q
80 INTEGER FIND
81 INTEGER MUL

```

! HOLDS BIT VALUES OF PREVIOUS LINE

! HOLDS WEIGHTS

! HOLDS RUNLENGTHS OF ENTIRE FILE

! HISTOGRAM DATA FOR 32 BITS IN ROW

! HISTOGRAM DATA FOR 64 BITS IN ROW

! HOLDS TOTALS FOR HISTOGRAMS

! WEIGHT COUNT FOR 32 BIT SEGMENTS

! WEIGHT COUNT FOR 64 BIT SEGMENTS

! DATA TO BE USED IN CALCULATIONS

! HOLDS VALUES OF NEIGHBORING BITS

! HOLDS VALUE OF PREVIOUS BIT

! NUMBER OF BITS IN FILE

! HOLDS POSITION OF 1 BIT IN CSL

! HOLDS WEIGHT OF EACH CSL

! CCITT FILE INFORMATION

! HOLDS VALUE OF WEIGHT OF FILE

! OUTPUT BUFFER

! PRESENT WORD IN INPUT BUFFER

! PRESENT WORD IN OUTPUT BUFFER

! INDEX OF CURRENT BIT IN LINE

! LOOP COUNTERS

! INDEX USED IN SEARCH

! NARROWING VARIABLE USED IN SEARCH

*****FILE PARAMETERS*****

```

82 C
83 C
84 C
85 CHARACTER INFILE*25
86 CHARACTER OTFILE*25
87 CHARACTER DATFIL*25
88 CHARACTER ACCTYP*25
89 INTEGER IRCL,JRCL,OREC
90 INTEGER ITBUF(15)
91 INTEGER TERM
92 INTEGER LPPFIL
93 INTEGER PELFIL
94 INTEGER OUTFIL
95 INTEGER DTFIL
96 INTEGER ITLOG
97 LOGICAL EXISTS

```

! FILE PARAMETERS
! ARRAY FOR TIME-OF-DAY
! TERMINAL LU
! PRINTER LU
! LU OF INPUT FILE
! LU OF OUTPUT FILE
! LU OF DATA FILE
! LIBRARY FUNCTION

DATA TERM,LPPFIL,PELFIL,OUTFIL,DTFIL/23,6,2,3,4/

```

102 C ***** INITIALIZING *****
103 C
104 C
105 DO I=-1,BITS
106   PREV(I)=0
107 END DO
108
109 DO I=0,BITS
110   WEIGHT(I)=0
111   RNLGTH(I)=0
112 END DO
113
114 DO I=0,32
115   THWT(I)=0
116 END DO
117
118 DO I=0,64
119   SXVT(I)=0
120 END DO
121 C ***** BEGIN PROGRAM *****
122 C
123 C
124 C GET INPUT FILE NAME
125 C
126 CALL RCPAR(1,INFILE)
127 INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
128   *EXIST=EXISTS, ACCESS=ACCTYP,IOSTAT=ISTAT)
129 IF(ISTAT.NE.0) STOP 'INQUIRE ERROR'
130 IF(EXISTS) THEN
131   IF(ACCTYP.EQ.'SEQUENTIAL') THEN
132     IF(IRCL.EQ.0.AND.MREC.EQ.0) THEN
133       OPEN(UNIT=PELFIL,FILE=INFILE,ACCESS=ACCTYP)
134       CALL LGBUF(LBUF,BUFD)
135       READ(UNIT=PELFIL,IOSTAT=ISTAT) (BUFFER(I),I=1,BUFD+1)
136       IF(ISTAT.NE.0) THEN
137         IRCL=-1*LOG(1)
138         MREC=MAXLIN
139         BACKSPACE(UNIT=PELFIL,IOSTAT=ISTAT)
140         IF(ISTAT.NE.0) STOP 'BACKSPACE ERROR'
141       ELSE
142         STOP 'NO ERROR IS AN ERROR'
143       END IF
144     ELSE
145       STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
146     END IF
147   ELSE IF(ACCTYP.EQ.'DIRECT') THEN
148     OPEN(UNIT=PELFIL,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT',
149   *RECL=IRCL,MAXREC=MREC)
150   ELSE
151     STOP 'ACCTYP UNDEFINED'
152   END IF
153 ELSE
154   STOP 'INPUT FILE DOES NOT EXIST'
155 END IF
156 C

```



```

157 C ***** READ IN INPUT PARAMETERS *****
158 C
159     BUFDIM=8
160     DO WHILE (BUFDIM.LT.1.OR.BUFDIM.GT.BUFD)
161         WRITE(TERM,*)((SENDER NUMBER OF WORDS PER OUTPUT RECORD: ' '))
162         READ(TERM,*) BUFDIM
163     END DO
164
165     LINMAX=8
166     DO WHILE (LINMAX.LT.1.OR.LINMAX.GT.MAXLIN)
167         WRITE(TERM,*)((NUMBER OF RECORDS TO BE OUTPUT - ' '))
168         READ(TERM,*) LINMAX
169     END DO
170
171     WRITE(TERM,*) 'CCITT FILE NUMBER?'
172     READ(TERM,*) FLNUM
173     WRITE(TERM,*) 'FILE RESOLUTION?'
174     READ(TERM,*) RESLTN
175 C ***** OPEN OUTPUT FILE *****
176 C
177 C
178     CALL RCPAR(3,OTFILE)
179     OREC=MIN(MREC,LINMAX)
180     JRCL=BUFDIM*2
181     OPEN(UNIT=OTFIL,FILE=OTFILE,ACCESS='DIRECT',RECL=JRCL
182           & ,MAXREC=OREC,STATUS='OLD',IOSTAT=ISTAT)
183     IF(ISTAT.NE.0) STOP 'OUTPUT FILE OPEN ERROR'
184
185     WRITE(TERM,*) 'NUMBER OF RECORDS IN INPUT FILE?'
186     READ(TERM,*) DERECS
187     MREC=MIN(MREC,DERECS)
188 C ***** WRITE INPUT PARAMETERS *****
189 C
190 C
191     CALL FTIME(ITBUF)
192     WRITE(LPFILE,((IHS,15A2))) ITBUF
193     WRITE(LPFILE,((INPUT FILE NAME - 'A2B'))) INFILE
194     WRITE(LPFILE,((OUTPUT FILE NAME - 'A2B'))) OTFILE
195     WRITE(LPFILE,2SS) FLNUM,RESLTN,BUFDIM,OREC,IRCL,MREC,JRCL
196     2SS FORMAT('INPUT PARAMETERS: /
197           * *****/
198           * FACSIMILE IMAGE: CCITT IMAGE #',12/
199           * RESOLUTION: ',14,' LINES PER INCH'/
200           * NUMBER OF WORDS PER OUTPUT RECORD =',16/
201           * NUMBER OF RECORDS TO BE WRITTEN =',16/
202           * INPUT RECORD SIZE =',17,' BYTES'/
203           * NO. OF INPUT RECORDS =',17/
204           * OUTPUT RECORD SIZE =',17,' BYTES')
205 C ***** READ IN PROBABILITY DATA *****
206 C
207 C
208     CALL RCPAR(4,DATFIL)
209     OPEN(UNIT=DTFIL,FILE=DATFIL)
210     DO 1-1,16
211         READ(DTFIL,*) PROB(1)

```

```

212      END DO
213
214      BUFMAX=IRCL/2
215      FIVGT=B
216
217 C***** BEGIN FILE PROCESSING *****
218 C
219 C
220      DO I=1,MREC
221
222          CT=1
223          PSOLD=B
224          LNWGT=B
225          SIXCT=B
226          THRCT=B
227          PREBIT=B
228
229          CALL LGBUF(LBUF,BUFD)
230          READ(UNIT=PELFI,IOSTAT=ISTAT) (BUFFER(P),P=1,BUFMAX)
231          IF(ISTAT.NE.B) THEN
232              WRITE(TERM,('INPUT READ ERROR, ISTAT=',I4)) ISTAT
233              STOP
234              END IF
235
236          DO K=1,BUFMAX
237              BITWRD=BUFFER(K)
238              OBTRD=B
239
240              DO L=1,B,-1
241                  IF (BTST(BITWRD,L))THEN
242                      BITVAL=1
243                  ELSE
244                      BITVAL=B
245                  END IF
246                  DO Q=1,3
247                      NBR(Q)=PREV(Q-1)+(CT-1)
248                  END DO
249                      NBR(4)=PREBIT
250                      PREV(CT-1)=PREBIT
251                      PREBIT=BITVAL
252 C***** FIND PROPER NEIGHBOR TEMPLATE *****
253 C
254      FIND=1
255      MUL=16
256      DO P=1,4
257          IF (NBR(P).EQ.1) THEN
258              FIND=FIND*MUL/2
259              END IF
260              MUL=MUL/2
261          END DO
262
263          IF (PROB(FIND).NE.BITVAL) THEN
264              OBTRD=IBSET(OBTRD,L)
265              LNWGT=LNWGT+1
266

```

```

267 SIXCT=SIXCT+1
268 THRCT=THRCT+1
269 RNLGTH(CT-PSHOLD)=RNLGTH(CT-PSHOLD)+1
270 PSHOLD=CT
271 END IF
272 IF (MOD(CT,32).EQ.0) THEN
273   THWT(THRCT)=THWT(THRCT)+1
274   THRCT=0
275 END IF
276 IF (MOD(CT,64).EQ.0) THEN
277   SIXT(SIXCT)=SIXT(SIXCT)+1
278   SIXCT=0
279 END IF
280 CT=CT+1
281 END DO
282 I NO BITS IN WORD
283 OBUFF(K)=OBTWRD
284
285 END DO
286 I NO WORDS IN RECORD
287
288 PREV(CT-1)=BITVAL
289 FIVGT=FIVGT+LNVGT
290 WEIGHT(LNVGT)=WEIGHT(LNVGT)+1
291 C***** WRITE RECORD TO OUTPUT FILE *****
292 C
293 CALL LGBUF(LBUF,BUFD)
294 WRITE(UNIT=OUTFIL,IOSTAT=ISTAT) (OBUFF(Q),Q=1,BUFDIM)
295 IF (ISTAT.NE.0) THEN
296   WRITE(TERM,('OUTPUT READ ERROR, ISTAT=',I4)) ISTAT
297   STOP
298 END IF
299
300 END DO
301 I RECORDS IN FILE
302 C***** PRINT HISTOGRAM *****
303 C
304 BITNUM=MREC*16*BUFMAX
305 WRITE(LPFILE,*)
306 WRITE(LPFILE,250) BITNUM,FIVGT
307 250 FORMAT ('NUMBER OF BITS IN CIF T=',I12/,
308           'WEIGHT OF CIF V=',I12//)
309 WRITE(LPFILE,*) 'HISTOGRAM OF WEIGHTS OF CONDITIONED SCAN'
310 WRITE(LPFILE,*) 'LINES WITHIN THE CONDITIONED IMAGE FILE'
311 WRITE(LPFILE,*)
312 WRITE(LPFILE,*) 'WEIGHT'
313 TOTAL=0
314 WRITE(LPFILE,*) '*****'
315
316 DO I=0,BITS
317   IF (WEIGHT(I).NE.0) THEN
318     WRITE(LPFILE,300) (I,WEIGHT(I))
319     TOTAL=TOTAL+WEIGHT(I)
320   END IF
321 END DO

```

```

322 WRITE(LPFIL,27#) TOTAL
323
324 WRITE(LPFIL,*) 'RUNLENGTHS IN CONDITIONED IMAGE FILE'
325 WRITE(LPFIL,*) ' '
326 WRITE(LPFIL,*) 'RUNLENGTHS OCCURRENCES'
327 TOTAL=0
328 WRITE(LPFIL,*) '*****'
329
330 DO I=1,BITS
331 IF (RNLGTH(I).NE.0) THEN
332 WRITE(LPFIL,30#) (I,RNLGTH(I))
333 TOTAL=TOTAL+RNLGTH(I)
334 END IF
335 END DO
336
337 WRITE(LPFIL,27#) TOTAL
338 WRITE(LPFIL,*) 'THIRTY TWO BIT SEGMENT HISTOGRAM'
339 WRITE(LPFIL,*) ' '
340 WRITE(LPFIL,*) 'WEIGHT OCCURRENCES'
341 TOTAL=0
342 WRITE(LPFIL,*) '*****'
343
344 DO I=0,32
345 IF (THWT(I).NE.0) THEN
346 WRITE(LPFIL,30#) (I,THWT(I))
347 TOTAL=TOTAL+THWT(I)
348 END IF
349 END DO
350
351 WRITE(LPFIL,27#) TOTAL
352 WRITE(LPFIL,*) 'SIXTY-FOUR BIT SEGMENT HISTOGRAM'
353 WRITE(LPFIL,*) ' '
354 WRITE(LPFIL,*) 'WEIGHT OCCURRENCES'
355 TOTAL=0
356 WRITE(LPFIL,*) '*****'
357
358 DO I=0,64
359 IF (SXWT(I).NE.0) THEN
360 WRITE(LPFIL,30#) (I,SXWT(I))
361 TOTAL=TOTAL+SXWT(I)
362 END IF
363 END DO
364
365 WRITE(LPFIL,27#) TOTAL
366 FORMAT ('-----',/
367 ' ' , TOTAL',119//
368 ' ' , '-----',/
369 30#) FORMAT (15,12#)
370
371 CLOSE(UNIT=PELFI)
372 CLOSE(UNIT=OUTFIL)
373 C*****
374 C
375 C
376 C END OF PROGRAM

```

Page 8 CREATE Opts: 77/LCVI Wed Aug 14, 1985 11:49:18 am
CIF Generation Program /SLDC/SLDCI/CREATE.FTN

377 C
378 C
379
380
 END

Module CREATE
FTN7X 2448/858394 No errors Prog: 28671 Blank Common: None
 No warnings Save: None Local Ems: None

```
3 $FILES 2.2
4
5 PROGRAM ENCODE
6
7 C
8 C
9 C AUTHOR J. P. DIMAGGIO
10 C
11 C DESCRIPTION:
12 C
13 C This program encodes a conditioned image file
14 C using SLDC encoding techniques. The user is
15 C prompted for all input parameters including
16 C file sizes, amount of file to process, input
17 C design parameters, and option to create an
18 C output file (and/or) print line by line
19 C compression statistics.
20 C
21 C
22 C
23 C CALLING SEQUENCE:
24 C
25 C RU,ENCODE,<INPUT NAMR>,<OUTPUT NAMR>
26 C
27 C
28 C INPUT PARAMETERS:
29 C NUMBER OF RECORDS PER OUTPUT FILE
30 C DECISION FOR OUTPUT FILE
31 C MAXIMUM WEIGHT PER SEGMENT
32 C CCITT FILE NUMBER
33 C FILE RESOLUTION
34 C ARITHMETIC WORD LENGTH
35 C NUMBER OF WORDS PER INPUT FILE
36 C NUMBER OF RECORDS IN INPUT FILE
37 C SEGMENT LENGTH TO BE COMPRESSED
38 C DECISION FOR LINE BY LINE COMPRESSION STATS
39 C
40 C
41 C
42 C
43 C
44 C INCLUDE ENCODE.INC
45 C
46 C IMPLICIT NONE
47 C
48 C
49 C DEFINE PARAMETERS
50 C
51 C INTEGER MAXLIN I MAXIMUM NO. OF OUTPUT RECORDS
52 C INTEGER DERECS I RECORDS IN INPUT FILE
53 C INTEGER BUFD I MAXIMUM NO. OF WORDS PER INPUT RECORD
54 C
55 C PARAMETER (MAXLIN=5000)
56 C
```

```

14+  PARAMETER (BUFD=225)
15+C *****
16+C *****
17+  INTEGER ISTAT      I I-O STATUS VARIABLE *****
18+  INTEGER LBUF(BUFD) I READ/WRITE BUFFER FOR FORTRAN 77
19+  INTEGER CSL(BUFD)  I CSL BUFFER
20+  INTEGER ESL(BUFD)  I ENCODED SCAN LINE BUFFER
21+C *****
22+  INTEGER LINMAX      I NO. OF RECORDS TO BE OUTPUT
23+  INTEGER BUFDIM      I NO. OF WORDS PER OUTPUT RECORD
24+  INTEGER*4 MREC      I NO. OF RECORDS IN INPUT FILE
25+C *****
26+  INTEGER BUFMAX      I INPUT RECORD SIZE IN WORDS
27+C *****
28+C *****
29+C *****
30+  INTEGER*4 WEIGHT(-1:34)
31+  INTEGER CODLEN(-1:34)
32+  INTEGER SEGBUF(BUFD)
33+  INTEGER MAXVGT
34+  INTEGER CTWRD
35+  INTEGER ESLPOS
36+  INTEGER CSLPOS
37+  REAL RATIO
38+  INTEGER NBPV
39+  INTEGER CHECK
40+  INTEGER*4 TOTAL
41+  INTEGER VRDS
42+  INTEGER SEGT
43+  INTEGER FLNUM
44+  INTEGER RESLTN
45+  INTEGER*4 CODE
46+  INTEGER SEGVGT
47+  INTEGER POSITN
48+  INTEGER*4 RANK
49+  INTEGER SGNES
50+  INTEGER*4 R1,R2
51+  INTEGER L1
52+  INTEGER LONES
53+  INTEGER LEN
54+  REAL LOG2
55+  INTEGER NUMBER
56+  INTEGER*4 BINOM
57+  INTEGER*4 LNWGT
58+  INTEGER THWGT
59+  INTEGER SEGL
60+  INTEGER BITWRD
61+  INTEGER CT
62+  INTEGER M.I.K.L.O
63+  CHARACTER DECIDE
64+  CHARACTER PRSTS
65+  *****
66+  *****
67+C *****
68+C *****

```

I HOLDS HUFFMAN CODES IN PROPER ORDER
 I HOLDS HUFF. CODE WEIGHTS IN PROP. ORDER
 I SEGMENT BUFFER
 I MAX WEIGHT ENCODABLE OFR SEGMENT
 I CURRENT WORD IN CSL
 I CURRENT POSITION IN ESL
 I CURRENT POSITION IN CSL
 I COMPRESSION RATIO
 I NUMBER OF BITS PER WORD
 I BIT HOLDER
 I TOTAL NUMBER OF BITS COMPRESSED
 I NUMBER OF WORDS IN SEGMENT
 I NUMBER OF SEGMENTS IN A LINE
 I CURRENT COUNT OF SEGMENTS IN A LINE
 I CCITT IMAGE FILE NUMBER
 I FILE RESOLUTION
 I HUFFMAN CODE OF SEGMENT WEIGHT
 I CURRENT SEGMENT WEIGHT
 I CURRENT POSITION IN SEGMENT
 I RANK OF SEGMENT
 I ONES SEEN IN SEGMENT
 I TEMPORARIES IN CASE OF 32 BIT WORD
 I
 I ONES SEEN IN LINE
 I LENGTH OF RANK
 I CONSTANT USED IN EQUATION
 I NUMBER OF BITS IN A LINE
 I BINOMIAL FUNCTION RETURN
 I HOLDS WEIGHT OF EACH CSL
 I SINGLE INTEGER VALUE FOR LNWGT
 I SEGMENT LENGTH TO BE COMPRESSED
 I PRESENT WORD IN INPUT BUFFER
 I INDEX OF CURRENT BIT IN LINE
 I LOOP COUNTERS
 I DECISION FLAG FOR OUTPUT FILE
 I DECISION FLAG FOR LINE BY LINE
 I CGMPRESSION STATISTICS

```

69+C CHARACTER INFILE*2#
70+C CHARACTER OTFILE*2#
71+C CHARACTER ACCTYP*2#
72+C INTEGER IRCL,JRCL,OREC      ! FILE PARAMETERS
73+C INTEGER ITBUF(15)          ! ARRAY FOR TIME-OF-DAY
74+C INTEGER TERM              ! TERMINAL LU
75+C INTEGER LPFIL             ! PRINTER LU
76+C INTEGER PELFIL            ! LU OF INPUT FILE
77+C INTEGER OUTFIL            ! LU OF OUTPUT FILE
78+C INTEGER ITLOG             ! LIBRARY FUNCTION
79+C
80+C LOGICAL EXISTS              ! TRUE IF SEG IS FINISHED ENCODING
81+C LOGICAL DONSEG             ! TRUE IF LINE IS FINISHED ENCODING
82+C LOGICAL DONLIN            ! FUNCTION USED TO COMPUTE BINOMIAL
83+C EXTERNAL BINOM             ! COEFFICIENT
84+C
85+C EXTERNAL MB48               ! DESCRIBED IN REPORT
86+C EXTERNAL MI28
87+C
88+C COMMON /ENCODE/WEIGHT,SEGL,NUMBER,CODLEN
89+C
90+C *****INITIALIZING*****
91+C
92+C DATA TERM,LPFIL,PELFIL,OUTFIL/1.6,2.3/
93+C
  
```



```

46 C .....
47 C .....
48 C .....
49 C .....
50 C .....
51 .....
52 DO L=0,33
53   WEIGHT(L)=0
54 END DO
55 LOG2=LOG(2.0)
56 .....
57 C GET INPUT FILE NAME
58 C .....
59 C .....
60 CALL RCPAR(1,INFILE)
61 INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
62   & EXIST=EXIST, ACCESS=ACCTYP,IOSTAT=ISTAT)
63 IF(ISTAT.NE.0) STOP 'INQUIRE ERROR'
64 IF(EXIST) THEN
65   IF(ACCTYP.EQ.'SEQUENTIAL') THEN
66     IF(IRCL.EQ.0.AND.MREC.EQ.0) THEN
67       OPEN(UNIT=PELFI,FILE=INFILE,ACCESS=ACCTYP)
68       CALL LGBUF(LBUF,BUFD)
69       READ(UNIT=PELFI,IOSTAT=ISTAT) (CSL(I),I=1,BUFD+1)
70       IF(ISTAT.NE.0) THEN
71         IRCL=ITLOG(
72           MREC-MAXLIN
73           BACKSPACE(UNIT=PELFI,IOSTAT=ISTAT)
74           IF(ISTAT.NE.0) STOP 'BACKSPACE ERROR'
75       ELSE STOP 'NO ERROR IS AN ERROR'
76       END IF
77     ELSE STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
78     END IF
79   ELSE IF(ACCTYP.EQ.'DIRECT') THEN
80     OPEN(UNIT=PELFI,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT',
81       & RECL=IRCL,MAXREC=MREC)
82   ELSE
83     STOP 'ACCTYP UNDEFINED'
84   END IF
85 ELSE
86   STOP 'INPUT FILE DOES NOT EXIST'
87 END IF
88 BUFDIM=IRCL/2
89 .....
90 .....
91 C Read number of records to be written
92 C .....
93 C .....
94 C .....
95 LINMAX=0
96 DO WHILE (LINMAX.LT.1.OR.LINMAX.GT.MAXLIN)
97   WRITE(TERM,*) 'HOW MANY RECORDS ARE TO BE PROCESSED?'
98   READ(TERM,*) LINMAX
99 END DO
100 DECIDE = 'X'
101

```

```

181 DO WHILE (DECIDE.NE.'Y'.AND.DECIDE.NE.'N')
182   WRITE(TERM,*) 'CREATE OUTPUT FILE? (Y or N)'
183   READ(TERM,*(A1)) DECIDE
184   END DO
185 C
186 C Get output file name and open if requested
187 C
188 IF (DECIDE.EQ.'Y') THEN
189   CALL RCPAR(2,OTFILE)
190   OREC=MIN(MREC,LIMAX)
191   JRCL=BUFDIM*2
192   OPEN(UNIT=OUTFIL,FILE=OTFILE,ACCESS='DIRECT',RECL=JRCL
193     & ,MAXREC=OREC,STATUS='NEW',IOSTAT=ISTAT)
194   IF (ISTAT.NE.0) THEN
195     WRITE(TERM,*) 'OUTPUT FILE OPEN ERROR',ISTAT
196     STOP
197   END IF
198 END IF
199 C
200 C Inquire for other input parameters
201 C
202 WRITE(TERM,*) 'MAXIMUM WEIGHT PER SEGMENT?'
203 READ(TERM,*) MAXVGT
204 WRITE(TERM,*) 'CCITT FILE NUMBER?'
205 READ(TERM,*) FLNUM
206 WRITE(TERM,*) 'RESOLUTION?'
207 READ(TERM,*) RESLTN
208 WRITE(TERM,*) 'ARITHMETIC WORD LENGTH?'
209 READ(TERM,*) MBPV
210 WRITE(TERM,*) 'WORDS PER INPUT RECORD?'
211 READ(TERM,*) BUFWAX
212 WRITE(TERM,*) 'NO. OF INPUT RECORDS?'
213 READ(TERM,*) DERECS
214 WRITE(TERM,*) 'SEGMENT LENGTH PARAMETER?'
215 READ(TERM,*) SEGL
216 MREC=MIN(MREC,DERECS)
217 C
218 C Write input parameters
219 C
220 WRITE(LPFIL,*) ' SLDC ENCODING PROGRAM '
221 WRITE(LPFIL,*) ' *****'
222 WRITE(LPFIL,*) ' '
223 CALL FTIME(ITBUF)
224 WRITE(LPFIL,*(1H$,15A2')) ITBUF
225 WRITE(LPFIL,*(1H$,15A2')) INFILE
226 IF (DECIDE.EQ.'Y') THEN
227   WRITE(LPFIL,*(1H$,15A2')) OTFILE
228 END IF
229 C
230 WRITE(LPFIL,2H$) JRCL,MREC,JRCL,OREC,MBPV,FLNUM,RESLTN
231 FORMAT(' INPUT PARAMETERS: '//
232   & ' *****'//
233   & ' INPUT RECORD SIZE =17' BYTES'/

```

```

156 *      * NO. OF INPUT RECORDS = '16/'
157 *      * OUTPUT RECORD SIZE = '16' BYTES '/'
158 *      * NO. OF OUTPUT RECORDS = '15/'
159 *      * NUMBER OF BITS PER WORD = '13'//
160 *      * COMPRESSED CIF FROM: CCITT IMAGE # '12/'
161 *      * RESOLUTION: '14' LINES PER INCH//
162
163
164 VRDS=SEGL/16
165 NUMBER=BUFMAX*16
166 SEGS=NUMBER/SEGL+1
167 WRITE(LPFIL,*) 'SEGMENT LENGTH BEING COMPRESSED= ',SEGL
168 WRITE(LPFIL,*) 'MAXIMUM WEIGHT ENCODABLE= ',MAXWGT
169
170 PRISTS = 'X'
171 DO WHILE(PRISTS.NE.'Y'.AND.PRISTS.NE.'N')
172   WRITE(TERM,*) 'LINE BY LINE COMPRESSION STATISTICS? (Y or N)'
173   READ(TERM,*(A1)) PRISTS
174   END DO
175
176 IF (PRISTS.EQ.'Y') THEN
177   WRITE(LPFIL,*) 'COMPRESSION STATISTICS FOR EACH LINE'
178   WRITE(LPFIL,*) '*****'
179   WRITE(LPFIL,*) 'LINE BITS COMPRESSION'
180   END IF
181
182 C***** BEGIN FILE ENCODING *****
183 C
184 TOTAL=0
185 DO Q=1,MREC
186   DO M=1,BUFMAX
187     ESL(M)=0
188     CSL(M)=0
189   END DO
190   CALL LGBUF(LBUF,BUFD)
191   READ(UNIT=PELFI,IOSTAT=ISTAT)(CSL(M),M=1,BUFMAX)
192   IF(ISTAT.NE.0) THEN
193     WRITE(TERM,*(INPUT READ ERROR ,ISTAT=,I4)) ISTAT
194     STOP
195   END IF
196   LNWGT=0
197   CSLPOS=1
198
199 C***** Calculate line weight *****
200 C
201   DO L=1,BUFMAX
202     BITWRD=CSL(L)
203     DO K=15,8,-1
204       IF(BTEST(BITWRD,K)) THEN
205         LNWGT=LNWGT+1
206       END IF
207     END DO
208   END DO
209   IF(LNWGT.EQ.0) THEN
210     ESL(1)=16
  
```

```

211 TOTAL=TOTAL+1
212 ESLPOS=1
213 ELSE
214 CALL MI28(LMWGT,ESL,1,13)
215 ESLPOS=14
216 TMWGT=LMWGT
217 CALL CODGEN(TMWGT,MAXWGT)
218 DONLIN=.FALSE.
219 CTWRD=1
220 LONES=0
221 SECT=0
222 DO WHILE (.NOT.DONLIN)
223 C==== Begin line encoding =====
224 C
225 CALL MB48(SEGBUF,1,SEGL,CSL,CSLPOS)
226 CSLPOS=CSLPOS+SEGL
227 SEGT=SEGT+1
228 SEGWGT=0
229 CT=0
230 M=0
231 DO WHILE(CT.LT.SEGL)
232 IF(MOD(CT,16).EQ.0) THEN
233 M=M+1
234 CHECK=SEGBUF(M)
235 K=16
236 END IF
237 K=K-1
238 CT=CT+1
239 IF(BTEST(CHECK,K)) THEN
240 SEGWGT=SEGWGT+1
241 END IF
242 END DO
243 LEN=CODLEN(SEGWGT)
244 CODE=WEIGHT(SEGWGT)
245 LONES=LONES+SEGWGT
246
247 C==== Encode line weight =====
248 C
249 IF((SEGWGT.GT.MAXWGT).OR.(SEGWGT.EQ.0)) THEN
250 IF(SEGWGT.EQ.0) THEN
251 CALL MI28(CODE,ESL,ESLPOS,LEN)
252 ESLPOS=ESLPOS+LEN
253 ELSE
254 CODE=WEIGHT(MAXWGT,1)
255 LEN=CODLEN(MAXWGT+1)
256 CALL MI28(CODE,ESL,ESLPOS,LEN)
257 ESLPOS=ESLPOS+LEN
258 CALL MB48(ESL,ESLPOS,SEGL,SEGBUF,1)
259 ESLPOS=ESLPOS+SEGL
260 END IF
261 ELSE
262 CALL MI28(CODE,ESL,ESLPOS,LEN)
263 ESLPOS=ESLPOS+LEN
264 DONSEG=.FALSE.
265

```

```

266 POSITN=SEGL
267 RANK=B
268 M=B
269 CT=16
270 SGONES=B
271 DO WHILE(.NOT.DONSEG)
272 C ***** Encode rank *****
273 C
274 C
275 M=M+1
276 CHECK=SEGBUF(M)
277 K=16
278 DO WHILE((K.GT.B).AND.(.NOT.DONSEG))
279 K=K-1
280 POSITN=POSITN-1
281 IF((SEGVT-SGONES).EQ.(POSITN+1)) THEN
282 DONSEG=.TRUE.
283 ELSE
284 IF(ISTEST(CHECK,K)) THEN
285 IF(POSITN.NE.B) THEN
286 RANK=RANK+BINOM(POSITN,(SEGVT-SGONES))
287 END IF
288 SGONES=SGONES+1
289 END IF
290 IF((SGONES.GE.SEGVT).OR.(POSITN.LE.B)) THEN
291 DONSEG=.TRUE.
292 END IF
293 END IF
294 END DO
295 END DO
296 LEN=NINT((LOG(REAL(BINOM(SEGL,SEGVT)))/
297 LOG2)+B.5)
298 IF ((MBPV.EQ.32).AND.(LEN.GT.16)) THEN
299 R1=B
300 L1=MOD(LEN,16)
301 CALL MVBITS(RANK,16,L1,R1,B)
302 LEN=LEN-L1
303 CALL M12B(R1,ESL,ESLPOS,L1)
304 ESLPOS=ESLPOS+L1
305 END IF
306 R2=RANK
307 CALL M12B(R2,ESL,ESLPOS,LEN)
308 ESLPOS=ESLPOS+LEN
309 END IF
310 IF((LNONES.GE.LNWGT).OR.(SEGCT.EQ.SEGS)) THEN
311 DONLIN=.TRUE.
312 END IF
313 END DO
314 TOTAL-TOTAL+(ESLPOS-1)
315 END IF
316 IF(DECIDE.EQ.'Y') THEN
317 CALL LGBUF(LBUF,BUFD)
318 WRITE(UNIT-OUTFIL,IOSTAT=1STAT) (ESL(M),M-1,BUFMAX)
319 END IF
320 IF (PRISTS.EQ.'Y') THEN

```

```

321      RATIO=REAL(NUMBER)/REAL(ESLPOS)
322      WRITE(1,310) Q,ESLPOS,RATIO
323      END IF
324      END DO
325      RATIO=MREC*BUFMAX*16/REAL(TOTAL)
326      CALL FTIME(ITBUF)
327      WRITE(LPFIL,*) ' FINISHED:'
328      WRITE(LPFIL,'(1H,15A2)') ITBUF
329      WRITE(LPFIL,350) RATIO
330      FORMAT(16,17,F12.2,'1')
331      FORMAT(/' COMPRESSION RATIO= ',F8.2,'11')
332      C*****
333      C*****
334      C*****
335      C*****
336      C*****
337      C*****
338      END
  
```

```

3 SUBROUTINE CODGEN(LNVGT,MAXVGT)
4
5 IMPLICIT NONE
6
7 C***** PARAMETER DEFINITION *****
8 C
9 C
10 INTEGER MAXVGT I MAXIMUM WEIGHT VALUE ENCODABLE
11 INTEGER LNVGT I CURRENT LINE WEIGHT
12 C
13 C*****
14 INTEGER*4 WEIGHT(-1:34) I HUFFMAN CODE WEIGHT ARRAY
15 REAL*8 PROB(-1:34) I DATA TO BE USED IN CALCULATIONS
16 INTEGER INFO(-1:34) I HOLDS WEIGHTS OF PROBABILITIES
17 REAL*8 NB I REAL VALUE OF BITS PER LINE
18 INTEGER NUMBPL I NUMBER OF BITS IN A LINE
19 INTEGER*4 BINOM I BINOMIAL FUNCTION RETURN
20 INTEGER*4 CODE(-1:34) I TABLE HOLDING HUFFMAN CODES
21 INTEGER CODLEN(-1:34) I CORRESPONDING CODE LENGTHS FOR CODES
22 INTEGER CDVLEN(-1:34) I CODE LENGTH ARRAY
23 REAL*8 P I PROBABILITY OF 1 ON THAT LINE
24 REAL*8 TEMP I USED IN SWAP IN SORT
25 INTEGER TEMP2 I
26 INTEGER SEGL I SEGMENT LENGTH TO BE COMPRESSED
27 REAL*8 SUM I USED IN COMPUTING SK
28 INTEGER MAXWT I TEMPORARY MAXVGT VARIABLE
29 INTEGER M,I,Q,J I LOOP COUNTERS
30
31 EXTERNAL BINOM I BINOMIAL FUNCTION
32
33 COMMON /HUFBLK/PROB,CODE,MAXWT,CDVLEN
34 COMMON /ENCOD/WEIGHT,SEGL,NUMBPL,CODLEN
35

```

```

37 C*****
38 C
39 C BEGIN SUBROUTINE
40 C*****
41 C*****
42 C
43 DO I=1,33
44 CODE(I) = 0
45 PROB(I) = 0
46
47 END DO
48 MAXVT = MAXVT
49 NB=REAL(NUMBPL)
50 SUM=0.0
51 IF (REAL(LNVGT).EQ.NB) THEN
52 P=0.9999
53 ELSE
54 P=REAL(LNVGT)/NB
55 END IF
56 DO J=1,MAXVT
57 PROB(J)=(BINOM(SEGL,J))*(P**J)*((1.0-P)**(SEGL-J))
58 INFO(J)=J
59 SUM=SUM+PROB(J)
60
61 END DO
62 IF ((1.0 - SUM).LT.0.001) THEN
63 PROB(MAXVT+1)=0.001
64 ELSE
65 PROB(MAXVT+1)=1.0-SUM
66 END IF
67 INFO(MAXVT+1)=MAXVT+1
68 WRITE(1,*)(PROB(M),M=1,MAXVT+1)
69 C*****SORT*****
70 C*****
71 C
72 DO M=1,MAXVT+1
73
74 DO Q=MAXVT+1,M+1,-1
75
76 IF (PROB(Q-1).LT.PROB(Q)) THEN
77 TEMP=PROB(Q-1)
78 TEMP2=INFO(Q-1)
79 PROB(Q-1)=PROB(Q)
80 INFO(Q-1)=INFO(Q)
81 PROB(Q)=TEMP
82 INFO(Q)=TEMP2
83 END IF
84
85 END DO
86
87 END DO
88 CALL HUFCD
89
90 DO M=1,MAXVT+1
91

```



```
92      CODLEN(INFO(M))=CDVLEN(M)
93      WEIGHT(INFO(M))=CODE(M)
94      END DO
95
96      RETURN
97 C*****
98 C
99 C      END OF SUBROUTINE
100 C
101 C*****
102      END
```

Module CODGEN
FTN7X 2448/8583B4

No errors	Prog: 382	Blank Common: None
No warnings	Save: None	Local Ems: None

```

3 *****
4
5 MODULE NAME: HUF COD
6
7
8
9 DESCRIPTION: This module generates Huffman codewords
10 and code lengths from probabilities
11 generated by CODGEN.
12
13 *****
14
15 SUBROUTINE HUF COD
16 IMPLICIT NONE
17 INCLUDE HUFFMAN.INC
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
10
```

```

21 *****
22 *
23 * OBJECT: To perform a left-to-right reduction of the Huffman
24 * coding table in order to locate the column positions
25 * that determine the variable-length codewords.
26 *
27 * FUNC:
28 * 1. For OL-1 columns in Huffman table
29 * 2. Add last two entries in column
30 * 3. Determine its rank in next column
31 * 4. Place it in next column at that rank
32 * 5. Mark the position in POSN( )
33 * 6. Go to next column
34 *****
35
36 TBLNDX=VGTNUM
37 DO N4=8,VGTNUM
38 CLMNDX=8
39 PR8TMP=PROB(TBLNDX)+PROB(TBLNDX+1)
40 DO WHILE (PR8TMP.LT.PROB(CLMNDX))
41 CLMNDX=CLMNDX+1
42 END DO
43 DO N5=TBLNDX,CLMNDX+1,-1
44 PROB(N5)=PROB(N5-1)
45 END DO
46 PROB(CLMNDX)=PR8TMP
47 POSN(N4)=CLMNDX
48 TBLNDX=TBLNDX-1
49
50 END DO
51

```

```

53 *
54 *
55 * OBJECT: To perform a right-to-left expansion of the Huffman
56 * coding table in order to assign the variable-length
57 * codewords based on the local conditional statistics
58 * of the image.
59 *
60 * INPUT: Column positions from the REDUCE routine.
61 *
62 * OUTPUT: Variable-length codewords for the neighbor conditions.
63 *
64 * FUNC:
65 *      1. For QL-1 columns in Huffman table
66 *      2. Add a bit to codeword at next column's POSN( )
67 *      3. Set new bit of last entry in column to ZERO
68 *      4. Set new bit of last entry in next column to ONE
69 *      5. Go to next column
70 *
71 *
72 *
73 *
74 *
75 *
76 *
77 *
78 *
79 *
80 *
81 *
82 *
83 *
84 *
85 *
86 *
87 *
88 *
89 *
90 *
91 *
92 *
93 *
94 *
95 *

```

```

TBLNDX=1
CODE( )=0
CODE(1)=1
CDVLEN( )=1
CDVLEN(1)=1
DO M4=VGTNUM, , -1
  DO M5=0, TBLNDX
    IF (POSN(M4).EQ.M5) THEN
      COTMP=CODE(POSN(M4))
      LENTMP=CDVLEN(POSN(M4))
      DO M6=POSN(M4), TBLNDX-1
        CODE(M6)=CODE(M5+1)
        CDVLEN(M6)=CDVLEN(M5+1)
      END DO
      CODE(TBLNDX)=2*COTMP
      CODE(TBLNDX+1)=2*COTMP+1
      CDVLEN(TBLNDX)=LENTMP+1
      CDVLEN(TBLNDX+1)=CDVLEN(TBLNDX)
    END IF
  END DO
END DO
TBLNDX=TBLNDX+1
END DO
END DO
RETURN
END

```

```

1  INTEGER*4 FUNCTION BINOM(N,M)
2  INTEGER M,N
3  REAL*6 T1,T2,T3,A
4  INTEGER I,B
5  A = 1.0
6  T1 = DBLE(N)
7  IF ((2 * M).GT.N) THEN
8    B = N - M
9  ELSE
10   B = M
11  END IF
12  DO I=B,B-1
13    T2 = DBLE(I)
14    A = (A * ((T1 - T2) / (T2 + 1.0)))
15  END DO
16  BINOM = DINT(A + 0.5)
17  RETURN
18  END

```

Module BINOM

FTN7X 2445/850304

No errors
No warningsProg: 106
Save: NoneBlank Common: None
Local Ema: None

```
2 SFILS 2.2
3 PROGRAM DECODE
4 *****
5 C
6 C AUTHOR J. P. DIMAGGIO
7 C
8 C DESCRIPTION:
9 C
10 C This program decodes the Encoded Conditioned Image File
11 C consisting of segment state words, back into the Conditioned
12 C Image File.
13 C
14 C
15 C
16 C
17 C CALLING SEQUENCE:
18 C
19 C RU.DECODE,<INPUT NAMR>,<OUTPUT NAMR>
20 C
21 C
22 C INPUT PARAMETERS:
23 C NUMBER OF WORDS PER OUTPUT RECORD
24 C NUMBER OF RECORDS PER OUTPUT FILE
25 C NUMBER OF WORDS IN INPUT FILE
26 C NUMBER OF RECORDS IN INPUT FILE
27 C SEGMENT LENGTH TO BE COMPRESSED
28 C MAXIMUM WEIGHT ENCODABLE FOR THAT SEGMENT
29 C
30 C
31 C
32 C
33 C INCLUDE DECODE.INC
34 C
35 C IMPLICIT NONE
36 C
37 C
38 C
39 C
40 C
41 C
42 C
43 C
44 C
45 C
46 C
47 C
48 C
49 C
50 C
51 C
52 C
53 C
54 C
55 C
56 C
57 C
58 C
59 C
60 C
61 C
62 C
63 C
64 C
65 C
66 C
67 C
68 C
69 C
70 C
71 C
72 C
73 C
74 C
75 C
76 C
77 C
78 C
79 C
80 C
81 C
82 C
83 C
84 C
85 C
86 C
87 C
88 C
89 C
90 C
91 C
92 C
93 C
94 C
95 C
96 C
97 C
98 C
99 C
100 C
101 C
102 C
103 C
104 C
105 C
106 C
107 C
108 C
109 C
110 C
111 C
112 C
113 C
114 C
115 C
116 C
117 C
118 C
119 C
120 C
121 C
122 C
123 C
124 C
125 C
126 C
127 C
128 C
129 C
130 C
131 C
132 C
133 C
134 C
135 C
136 C
137 C
138 C
139 C
140 C
141 C
142 C
143 C
144 C
145 C
146 C
147 C
148 C
149 C
150 C
151 C
152 C
153 C
154 C
155 C
156 C
157 C
158 C
159 C
160 C
161 C
162 C
163 C
164 C
165 C
166 C
167 C
168 C
169 C
170 C
171 C
172 C
173 C
174 C
175 C
176 C
177 C
178 C
179 C
180 C
181 C
182 C
183 C
184 C
185 C
186 C
187 C
188 C
189 C
190 C
191 C
192 C
193 C
194 C
195 C
196 C
197 C
198 C
199 C
200 C
201 C
202 C
203 C
204 C
205 C
206 C
207 C
208 C
209 C
210 C
211 C
212 C
213 C
214 C
215 C
216 C
217 C
218 C
219 C
220 C
221 C
222 C
223 C
224 C
225 C
226 C
227 C
228 C
229 C
230 C
231 C
232 C
233 C
234 C
235 C
236 C
237 C
238 C
239 C
240 C
241 C
242 C
243 C
244 C
245 C
246 C
247 C
248 C
249 C
250 C
251 C
252 C
253 C
254 C
255 C
256 C
257 C
258 C
259 C
260 C
261 C
262 C
263 C
264 C
265 C
266 C
267 C
268 C
269 C
270 C
271 C
272 C
273 C
274 C
275 C
276 C
277 C
278 C
279 C
280 C
281 C
282 C
283 C
284 C
285 C
286 C
287 C
288 C
289 C
290 C
291 C
292 C
293 C
294 C
295 C
296 C
297 C
298 C
299 C
300 C
301 C
302 C
303 C
304 C
305 C
306 C
307 C
308 C
309 C
310 C
311 C
312 C
313 C
314 C
315 C
316 C
317 C
318 C
319 C
320 C
321 C
322 C
323 C
324 C
325 C
326 C
327 C
328 C
329 C
330 C
331 C
332 C
333 C
334 C
335 C
336 C
337 C
338 C
339 C
340 C
341 C
342 C
343 C
344 C
345 C
346 C
347 C
348 C
349 C
350 C
351 C
352 C
353 C
354 C
355 C
356 C
357 C
358 C
359 C
360 C
361 C
362 C
363 C
364 C
365 C
366 C
367 C
368 C
369 C
370 C
371 C
372 C
373 C
374 C
375 C
376 C
377 C
378 C
379 C
380 C
381 C
382 C
383 C
384 C
385 C
386 C
387 C
388 C
389 C
390 C
391 C
392 C
393 C
394 C
395 C
396 C
397 C
398 C
399 C
400 C
401 C
402 C
403 C
404 C
405 C
406 C
407 C
408 C
409 C
410 C
411 C
412 C
413 C
414 C
415 C
416 C
417 C
418 C
419 C
420 C
421 C
422 C
423 C
424 C
425 C
426 C
427 C
428 C
429 C
430 C
431 C
432 C
433 C
434 C
435 C
436 C
437 C
438 C
439 C
440 C
441 C
442 C
443 C
444 C
445 C
446 C
447 C
448 C
449 C
450 C
451 C
452 C
453 C
454 C
455 C
456 C
457 C
458 C
459 C
460 C
461 C
462 C
463 C
464 C
465 C
466 C
467 C
468 C
469 C
470 C
471 C
472 C
473 C
474 C
475 C
476 C
477 C
478 C
479 C
480 C
481 C
482 C
483 C
484 C
485 C
486 C
487 C
488 C
489 C
490 C
491 C
492 C
493 C
494 C
495 C
496 C
497 C
498 C
499 C
500 C
501 C
502 C
503 C
504 C
505 C
506 C
507 C
508 C
509 C
510 C
511 C
512 C
513 C
514 C
515 C
516 C
517 C
518 C
519 C
520 C
521 C
522 C
523 C
524 C
525 C
526 C
527 C
528 C
529 C
530 C
531 C
532 C
533 C
534 C
535 C
536 C
537 C
538 C
539 C
540 C
541 C
542 C
543 C
544 C
545 C
546 C
547 C
548 C
549 C
550 C
551 C
552 C
553 C
554 C
555 C
556 C
557 C
558 C
559 C
560 C
561 C
562 C
563 C
564 C
565 C
566 C
567 C
568 C
569 C
570 C
571 C
572 C
573 C
574 C
575 C
576 C
577 C
578 C
579 C
580 C
581 C
582 C
583 C
584 C
585 C
586 C
587 C
588 C
589 C
590 C
591 C
592 C
593 C
594 C
595 C
596 C
597 C
598 C
599 C
600 C
601 C
602 C
603 C
604 C
605 C
606 C
607 C
608 C
609 C
610 C
611 C
612 C
613 C
614 C
615 C
616 C
617 C
618 C
619 C
620 C
621 C
622 C
623 C
624 C
625 C
626 C
627 C
628 C
629 C
630 C
631 C
632 C
633 C
634 C
635 C
636 C
637 C
638 C
639 C
640 C
641 C
642 C
643 C
644 C
645 C
646 C
647 C
648 C
649 C
650 C
651 C
652 C
653 C
654 C
655 C
656 C
657 C
658 C
659 C
660 C
661 C
662 C
663 C
664 C
665 C
666 C
667 C
668 C
669 C
670 C
671 C
672 C
673 C
674 C
675 C
676 C
677 C
678 C
679 C
680 C
681 C
682 C
683 C
684 C
685 C
686 C
687 C
688 C
689 C
690 C
691 C
692 C
693 C
694 C
695 C
696 C
697 C
698 C
699 C
700 C
701 C
702 C
703 C
704 C
705 C
706 C
707 C
708 C
709 C
710 C
711 C
712 C
713 C
714 C
715 C
716 C
717 C
718 C
719 C
720 C
721 C
722 C
723 C
724 C
725 C
726 C
727 C
728 C
729 C
730 C
731 C
732 C
733 C
734 C
735 C
736 C
737 C
738 C
739 C
740 C
741 C
742 C
743 C
744 C
745 C
746 C
747 C
748 C
749 C
750 C
751 C
752 C
753 C
754 C
755 C
756 C
757 C
758 C
759 C
760 C
761 C
762 C
763 C
764 C
765 C
766 C
767 C
768 C
769 C
770 C
771 C
772 C
773 C
774 C
775 C
776 C
777 C
778 C
779 C
780 C
781 C
782 C
783 C
784 C
785 C
786 C
787 C
788 C
789 C
790 C
791 C
792 C
793 C
794 C
795 C
796 C
797 C
798 C
799 C
800 C
801 C
802 C
803 C
804 C
805 C
806 C
807 C
808 C
809 C
810 C
811 C
812 C
813 C
814 C
815 C
816 C
817 C
818 C
819 C
820 C
821 C
822 C
823 C
824 C
825 C
826 C
827 C
828 C
829 C
830 C
831 C
832 C
833 C
834 C
835 C
836 C
837 C
838 C
839 C
840 C
841 C
842 C
843 C
844 C
845 C
846 C
847 C
848 C
849 C
850 C
851 C
852 C
853 C
854 C
855 C
856 C
857 C
858 C
859 C
860 C
861 C
862 C
863 C
864 C
865 C
866 C
867 C
868 C
869 C
870 C
871 C
872 C
873 C
874 C
875 C
876 C
877 C
878 C
879 C
880 C
881 C
882 C
883 C
884 C
885 C
886 C
887 C
888 C
889 C
890 C
891 C
892 C
893 C
894 C
895 C
896 C
897 C
898 C
899 C
900 C
901 C
902 C
903 C
904 C
905 C
906 C
907 C
908 C
909 C
910 C
911 C
912 C
913 C
914 C
915 C
916 C
917 C
918 C
919 C
920 C
921 C
922 C
923 C
924 C
925 C
926 C
927 C
928 C
929 C
930 C
931 C
932 C
933 C
934 C
935 C
936 C
937 C
938 C
939 C
940 C
941 C
942 C
943 C
944 C
945 C
946 C
947 C
948 C
949 C
950 C
951 C
952 C
953 C
954 C
955 C
956 C
957 C
958 C
959 C
960 C
961 C
962 C
963 C
964 C
965 C
966 C
967 C
968 C
969 C
970 C
971 C
972 C
973 C
974 C
975 C
976 C
977 C
978 C
979 C
980 C
981 C
982 C
983 C
984 C
985 C
986 C
987 C
988 C
989 C
990 C
991 C
992 C
993 C
994 C
995 C
996 C
997 C
998 C
999 C
1000 C
```

```

25+ INTEGER MREC I NO. OF RECORDS IN INPUT FILE
26+ INTEGER BUFMAX I INPUT RECORD SIZE IN WORDS
28+C----- LOCAL VARIABLES -----
29+C-----
30+C-----
31+ INTEGER*4 WEIGHT(-1:34) I HOLDS WEIGHTS
32+ INTEGER CODELEN(-1:34) I HOLDS HUFF. CODE LENGTHS
33+ INTEGER SEGBUF(80) I SEGMENT BUFFER
34+ INTEGER MAXWGT I MAX WEIGHT ENCODABLE OFR SEGMENT
35+ INTEGER CTWRD I CURRENT WORD IN ESL
36+ INTEGER CTCOD I CURRENT CODE IN CODE SEARCH
37+ INTEGER CSLPOS I CURRENT POSITION IN CSL
38+ INTEGER ESLPOS I CURRENT POSITION IN ESL
39+ INTEGER*4 SUB I USED IN RANK DECODING PROCESS
40+ INTEGER POS I USED IN RANK DECODING PROCESS
41+ INTEGER RANLEN I LENGTH OF RANK
42+ INTEGER*4 TOTAL I TOTAL NUMBER OF BITS COMPRESSED
43+ REAL RATIO I COMPRESSION RATIO
44+ INTEGER VRDS I NUMBER OF WORDS IN SEGMENT
45+ INTEGER SEGS.SECT I SEGMENTS IN A LINE AND SEGMENT COUNTER
46+ INTEGER CHECK I WORD HOLDER FOR SEGBUF
47+ INTEGER SEGWGT I CURRENT SEGMENT WEIGHT
48+ INTEGER*4 RANK I RANK OF SEGMENT
49+ INTEGER LNMES I ONES SEEN IN LINE
50+ INTEGER*4 R1 I INTERMEDIATE RANK VALUE
51+ INTEGER L1 I INTERMEDIATE RANK LENGTH
52+ INTEGER MBPW I NUMBER OF BITS PER WORD
53+ REAL LOG2 I CONSTANT USED IN EQUATION
54+ INTEGER NUMBER I NUMBER OF BITS IN A LINE
55+ INTEGER*4 BINOM I BINOMIAL FUNCTION RETURN
56+ INTEGER LNWGT I HOLDS WEIGHT OF EACH CSL
57+ INTEGER SEGL I SEGMENT LENGTH TO BE COMPRESSED
58+ INTEGER SMLEN I SINGLE REP OF SEGL
59+ INTEGER*4 BITWRD I PRESENT WORD IN INPUT BUFFER
60+ INTEGER OUTWRD I USED IN RANK DECODING
61+ INTEGER CT I INDEX OF CURRENT BIT IN LINE
62+ INTEGER M,I,K,L I LOOP COUNTERS
63+C-----
64+C----- FILE PARAMETERS -----
65+C-----
66+ CHARACTER INFILE*20 I FILE PARAMETERS
67+ CHARACTER OTFILE*20 I ARRAY FOR TIME-OF-DAY
68+ CHARACTER ACCTYP*20 I TERMINAL LU
69+ INTEGER IRCL,JRCL,OREC I PRINTER LU
70+ INTEGER ITBUF(15) I LU OF INPUT FILE
71+ INTEGER TERM I LU OF OUTPUT FILE
72+ INTEGER LPFIL I LIBRARY ROUTINE
73+ INTEGER PELFIL I LIBRARY FUNCTION
74+ INTEGER OUTFIL
75+ INTEGER*4 I48
76+ INTEGER ITLOG
77+ LOGICAL EXISTS
78+ LOGICAL DNRANK
79+ I TRUE IF RANK DECODING IS FINISHED

```

80+	LOGICAL	DONE	
81+	LOGICAL	FOUND	
82+	EXTERNAL	BINOM	
83+C			! FUNCTION FOR COMPUTING BINOMIAL
84+	EXTERNAL	MB48	COEFFICIENT
85+	EXTERNAL	148	! DESCRIBED IN REPORT
86+			!
87+	COMMON	/ENCOD/VEIGHT,SEGL,NUMBER,CODLEN	
88+			
89+			
90+			

DATA TERM,LPFIL,PELFIL,OUTFIL/1.6.2.3/


```

34 C ***** BEGIN PROGRAM *****
35 C
36 C LOG2=LOG(2.B)
37 C
38 C
39 C GET INPUT FILE NAME
40 C
41 C CALL RCPAR(1,INFILE)
42 C INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
43 C & EXIST=EXISTS, ACCESS=ACCTYP,IOSTAT=ISTAT)
44 C IF(ISTAT.NE.B) STOP 'INQUIRE ERROR'
45 C IF(EXISTS) THEN
46 C IF(ACCTYP.EQ.'SEQUENTIAL') THEN
47 C IF(IRCL.EQ.B.AND.MREC.EQ.B) THEN
48 C OPEN(UNIT=PELFI,FILE=INFILE,ACCESS=ACCTYP)
49 C CALL LGBUF(LBUF,BUFD)
50 C READ(UNIT=PELFI,IOSTAT=ISTAT) (CSL(1),I=1,BUFD+1)
51 C IF(ISTAT.NE.B) THEN
52 C IRCL=ITLOG(I)
53 C MREC=MAXLIN
54 C BACKSPACE(UNIT=PELFI,IOSTAT=ISTAT)
55 C IF(ISTAT.NE.B) STOP 'BACKSPACE ERROR'
56 C ELSE
57 C STOP 'NO ERROR IS AN ERROR'
58 C END IF
59 C ELSE
60 C STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
61 C END IF
62 C ELSE IF(ACCTYP.EQ.'DIRECT') THEN
63 C OPEN(UNIT=PELFI,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT',
64 C & .RECL=IRCL,MAXREC=MREC)
65 C ELSE
66 C STOP 'ACCTYP UNDEFINED'
67 C END IF
68 C ELSE
69 C STOP 'INPUT FILE DOES NOT EXIST'
70 C END IF
71 C
72 C ***** READ IN INPUT PARAMETERS *****
73 C
74 C BUFDIM=B
75 C DO WHILE (BUFDIM.LT.1.OR.BUFDIM.GT.BUFD)
76 C WRITE(TERM,*)((SENTER NUMBER OF WORDS PER OUTPUT RECORD: '''))
77 C READ(TERM,*) BUFDIM
78 C END DO
79 C LINMAX=B
80 C DO WHILE (LINMAX.LT.1.OR.LINMAX.GT.MAXLIN)
81 C WRITE(TERM,*)((NUMBER OF RECORDS TO BE OUTPUT = '''))
82 C READ(TERM,*) LINMAX
83 C END DO
84 C ***** GET OUTPUT FILE NAME AND OPEN *****
85 C
86 C
87 C
88 C

```

```

89 CALL RCPAR(2,OTFILE)
90 OREC=MIN(MREC,LINMAX)
91 JRCL=BUFDIM*2
92 OPEN(UNIT=OUTFIL,FILE=OTFILE,ACCESS='DIRECT',RECL=JRCL
93 & ,MAXREC=OREC,STATUS='NEW',IOSTAT=ISTAT)
94 IF(ISTAT.NE.0) THEN
95 WRITE(TERM,*) 'OUTPUT FILE OPEN ERROR',ISTAT
96 END IF
97
98 WRITE(TERM,*) 'NUMBER OF BITS PER WORD?'
99 READ(TERM,*) NBPW
100 C***** WRITE INPUT PARAMETERS *****
101 C
102 C
103 CALL FTIME(ITBUF)
104 WRITE(LPFIL,('(1H,15A2)')) ITBUF
105 WRITE(LPFIL,('(1H,15A2)')) INFILE
106 WRITE(LPFIL,('(1H,15A2)')) OTFILE
107
108 WRITE(LPFIL,200) BUFDIM,OREC,IRCL,MREC,JRCL,NBPW
109 FORMAT('INPUT PARAMETERS:')
110 *
111 * NUMBER OF WORDS PER OUTPUT RECORD =',16/
112 * INPUT RECORD SIZE =',17' BYTES'
113 * NO. OF INPUT RECORDS =',17/
114 * OUTPUT RECORD SIZE =',17' BYTES'
115 * NUMBER OF BITS PER WORD =',17)
116
117
118 WRITE(TERM,*) 'WORDS PER INPUT RECORD?'
119 READ(TERM,*) BUFMAX
120 NUMBER=BUFMAX*16
121 WRITE(TERM,*) 'NUMBER OF RECORDS IN INPUT FILE?'
122 READ(TERM,*) DERECS
123 MREC=MIN(MREC,DERECS)
124 WRITE(TERM,*) 'SEGMENT LENGTH TO BE COMPRESSED?'
125 READ(TERM,*) SEGL
126 WRITE(TERM,*) 'MAX WEIGHT ENCODABLE?'
127 READ(TERM,*) MAXWGT
128 C
129 C***** GET RECORDS TO PROCESS *****
130 C
131 VRDS=SEGL/16+1
132 WRITE(LPFIL,*) 'SEGMENT LENGTH COMPRESSED= ',SEGL
133 WRITE(LPFIL,*) 'MAX WEIGHT= ',MAXWGT
134 SMLEN=SEGL
135 SEGS=NUMBER/SMLEN+1
136 TOTAL=0
137 DO I=1,MREC
138 CALL LGBUF(LBUF,BUFD)
139 READ(UNIT=PELFIL,IOSTAT=ISTAT)(ESL(M),M=1,BUFMAX)
140 DO L=1,BUFMAX
141 CSL(L)=0
142 END DO
143 IF (.NOT.(BTEST(ESL(1),15))) THEN

```

```

144 LNVGT=IBITS(ESL(1),3,13)
145 CALL CODGEN(LNVGT,MAXVGT)
146 DONE=.FALSE.
147 CSLPOS=1
148 SGCT=#
149 LNMES=#
150 CTWRD=1
151 ESLPOS=13
152 DO WHILE(.NOT.DONE)
153   FOUND=.FALSE.
154   SGCT=SGCT+1
155 C
156 C***** SEARCH FOR HUFFMAN CODE *****
157 C
158 BITWRD=148(ESL,ESLPOS+1,32)
159 CTCOD=#
160 DO WHILE(.NOT.FOUND)
161   IF ((IBITS(BITWRD,32-CODLEN(CTCOD),CODLEN(CTCOD)))
162     .EQ.(IBITS(WEIGHT(CTCOD),# ,CODLEN(CTCOD)))) THEN
163     &
164     FOUND=.TRUE.
165     SEGVGT=CTCOD
166   ELSE
167     CTCOD=CTCOD+1
168   END IF
169 END DO
170 ESLPOS=ESLPOS+CODLEN(CTCOD)
171 IF((SEGVGT.EQ.#).OR.(SEGVGT.EQ.MAXVGT+1)) THEN
172   IF (SEGVGT.EQ.#) THEN
173     DO L=1,VRDS
174       SEGBUF(L)=#
175     END DO
176   ELSE
177     CALL MB4B(SEGBUF,1,SMLEN,ESL,ESLPOS+1)
178     ESLPOS=ESLPOS+SMLEN
179     M=#
180     CT=#
181     DO WHILE(CT.LT.SMLEN)
182       IF(MOD(CT,16).EQ.#) THEN
183         M=M+1
184         CHECK=SEGBUF(M)
185         K=16
186       END IF
187       K=K-1
188       CT=CT+1
189       IF(BTEST(CHECK,K)) THEN
190         LNMES=LNMES+1
191       END IF
192     END DO
193     END IF
194   ELSE
195     LNMES=LNMES+SEGVGT
196     RANLEN=NINT((LOG(REAL(BINOM(SEGL,SEGVGT)))
197       /LOG2)+#.5)
198     IF ((MBPV.EQ.32).AND.(RANLEN.GT.16)) THEN

```

```

199 R1=M
200 L1=MOD(RANLEN,16)
201 R1=148(ESL,ESLPOS+1,L1)
202 CALL MVBITS(R1,M,L1,RANK,16)
203 RANLEN=RANLEN-L1
204 ESLPOS=ESLPOS+L1
205 END IF
206 R1=148(ESL,ESLPOS+1,RANLEN)
207 CALL MVBITS(R1,M,RANLEN,RANK,M)
208 ESLPOS=ESLPOS+RANLEN
209 CT=16
210 POS=SEGL
211 OUTVRD=M
212 DNRANK=.FALSE.
213 M=M
214 DO WHILE((POS.GT.M).AND.(.NOT.DNRANK))
215 POS=POS-1
216 CT=CT-1
217 IF(SEGVT.EQ.(POS+1)) THEN
218 DNRANK=.TRUE.
219 DO L=POS,M-1
220 IF(CT.LT.M) THEN
221 M=M+1
222 SEGUF(M)=OUTVRD
223 OUTVRD=M
224 CT=16
225 END IF
226 OUTVRD=IBSET(OUTVRD,CT)
227 CT=CT-1
228 END DO
229 ELSE
230 SUB=BINOM(POS,SEGVT)
231 IF ((RANK-SUB).GE.M) THEN
232 RANK=RANK-SUB
233 SEGVT=SEGVT-1
234 OUTVRD=IBSET(OUTVRD,CT)
235 END IF
236 END IF
237 IF(CT.LE.M) THEN
238 M=M+1
239 SEGUF(M)=OUTVRD
240 OUTVRD=M
241 CT=16
242 END IF
243 END DO
244 M=M+1
245 SEGUF(M)=OUTVRD
246 END IF
247 CALL MB48(CSL,CSLPOS,SMLEN,SEGUF,1)
248 CSLPOS=CSLPOS+SMLEN
249 IF((LNONES.GE.LNVT).OR.(SEGS.EQ.SGCT)) THEN
250 DONE=.TRUE.
251 END IF
252 END DO
253 TOTAL=TOTAL+ESLPOS

```

```

254 ELSE
255 TOTAL=TOTAL+1
256 END IF
257 CALL LGBUF(LBUF,BUFD)
258 WRITE(UNIT=OUTFIL,IOSTAT=ISTAT) (CSL(M),M=1,BUFMAX)
259 END DO
260 RATIO=REAL(NUMBER)*MREC/REAL(TOTAL)
261 WRITE(LPFIL,*) 'COMPRESSION RATIO= ',RATIO
262 268 FORMAT(15(115))
263
264 CLOSE(UNIT=PELFIL)
265 CLOSE(UNIT=OUTFIL)
266
267 C=====
268 C
269 C END OF PROGRAM
270 C
271 C=====
272
273 END

```

Module DECODE FTN7X 2445/855354 No errors No warnings Prog: 2881 Save: None Blank Common: None Local Ema: None

```

3 SFILS 3,4,5
4 PROGRAM RESTORE
5
6
7 C
8
9 C
10 C
11 C
12 C
13 C
14 C
15 C
16 C
17 C
18 C
19 C
20 C
21 C
22 C
23 C
24 C
25 C
26 C
27 C
28 C
29 C
30 C
31 C
32 C
33 C
34 C
35 C
36 C
37 C
38 C
39 C
40 C
41 C
42 C
43 C
44 C
45 C
46 C
47 C
48 C
49 C
50 C
51 C
52 C
53 C
54 C
55 C
56 C
57 C

*****
AUTHOR J. P. DIMAGGIO
PROGRAM LAST CHANGED <855814.1145>
DESCRIPTION:
This program restores the Conditioned Image File
back to the original file after decoding.

CALLING SEQUENCE:
RU,RESTORE,<INPUT NAMR>,<OUTPUT NAMR>,<PROBABILITY FILE>

*****
INPUT PARAMETERS:
NUMBER OF WORDS PER OUTPUT RECORD
NUMBER OF RECORDS PER OUTPUT FILE
NUMBER OF RECORDS IN INPUT FILE

*****
IMPLICIT NONE

*****
DEFINE PARAMETERS
INTEGER MAXLIN I MAXIMUM NO. OF OUTPUT RECORDS
INTEGER DERECs I RECORDS IN INPUT FILE
INTEGER BUFD I MAXIMUM NO. OF WORDS PER INPUT RECORD
INTEGER BITS I MAXIMUM NO. OF BITS PER RECORD RECORD
PARAMETER (MAXLIN=5000)
PARAMETER (BITS=4100)
PARAMETER (BUFD=225)

***** FILE DEFINITIONS *****
INTEGER ISTAT I I-O STATUS VARIABLE
INTEGER LBUF(BUFD) I READ/WRITE BUFFER FOR FORTRAN 77
INTEGER BUFR(BUFD) I RECORD BUFFER
INTEGER OBUFR(BUFD) I OUTPUT BUFFER

INTEGER LINMAX I NO. OF RECORDS TO BE OUTPUT
INTEGER BUFDIM I NO. OF WORDS PER OUTPUT RECORD
INTEGER MREC I NO. OF RECORDS IN INPUT FILE

INTEGER BUFRMAX I INPUT RECORD SIZE IN WORDS

```

```

58 C ***** LOCAL VARIABLES *****
59 C
60 C
61 C
62 C
63 C
64 C
65 C
66 C
67 C
68 C
69 C
70 C
71 C
72 C ***** FILE PARAMETERS *****
73 C
74 C
75 C
76 C
77 C
78 C
79 C
80 C
81 C
82 C
83 C
84 C
85 C
86 C
87 C
88 C
89 C
90 C

INTEGER PREV(-1:BITS)      ! HOLDS BIT VALUES OF PREVIOUS LINE
INTEGER PROB(16)           ! DATA TO BE USED IN CALCULATIONS
INTEGER NBR(4)             ! HOLDS VALUES OF NEIGHBORING BITS
INTEGER BITVAL             ! HOLDS VALUE OF CURRENT BIT
INTEGER PREBIT             ! HOLDS VALUE OF PREVIOUS BIT
INTEGER PSHOLD             ! HOLDS POSITION OF 1 BIT IN CSL
INTEGER BITWRD             ! PRESENT WORD IN INPUT BUFFER
INTEGER OBTWRD             ! PRESENT WORD IN OUTPUT BUFFER
INTEGER CT                 ! INDEX OF CURRENT BIT IN LINE
INTEGER P.I.K.L.O         ! LOOP COUNTERS
INTEGER FIND.MUL           ! NARROWING VARIABLES IN SEARCH

CHARACTER INFILE*20
CHARACTER DTFILE*20
CHARACTER DATFIL*20
CHARACTER ACCTVP*20
INTEGER IRCL,IRCL,OREC    ! FILE PARAMETERS
INTEGER ITBUF(15)         ! ARRAY FOR TIME-OF-DAY
INTEGER TERM              ! TERMINAL LU
INTEGER LPFIL             ! PRINTER LU
INTEGER PELFIL            ! LU OF INPUT FILE
INTEGER OUTFIL            ! LU OF OUTPUT FILE
INTEGER DTFIL             ! LU OF DATA FILE
INTEGER ITLOG             ! LIBRARY FUNCTION

LOGICAL EXISTS

DATA TERM,LPFIL,PELFIL,OUTFIL,DTFIL/1,6,2,3,4/

```

```

92 C ***** INITIALIZING *****
93 C
94 C
95 DO I=-1,BITS
96   PREV(I)=#
97 END DO
98 C ***** BEGIN PROGRAM *****
99 C
100 C
101 CALL RCPAR(1,INFILE)
102 INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
103   & EXIST=EXISTS, ACCESS=ACCTYP,IOSTAT=ISTAT)
104 IF(ISTAT.NE.#) STOP 'INQUIRE ERROR'
105 IF(EXISTS) THEN
106   IF(ACCTYP.EQ.'SEQUENTIAL') THEN
107     IF(IRCL.EQ.#.AND.MREC.EQ.#) THEN
108       OPEN(UNIT=PELFIL,FILE=INFILE,ACCESS=ACCTYP)
109       CALL LGBUF(LBUF,BUFD)
110       READ(UNIT=PELFIL,IOSTAT=ISTAT) (BUFFER(I),I=1,BUFD+1)
111       IF(ISTAT.NE.#) THEN
112         IRCL=ITLOGI
113         MREC=MAXLIN
114         BACKSPACE(UNIT=PELFIL,IOSTAT=ISTAT)
115         IF(ISTAT.NE.#) STOP 'BACKSPACE ERROR'
116       ELSE
117         STOP 'NO ERROR IS AN ERROR'
118       END IF
119     ELSE
120       STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
121     END IF
122   ELSE IF(ACCTYP.EQ.'DIRECT') THEN
123     OPEN(UNIT=PELFIL,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT',
124   & ,RECL=IRCL,MAXREC=MREC)
125   ELSE
126     STOP 'ACCTYP UNDEFINED'
127   END IF
128 ELSE
129   STOP 'INPUT FILE DOES NOT EXIST'
130 END IF
131 C ***** READ IN INPUT PARAMETERS *****
132 C
133 C
134 BUFDIM=#
135 DO WHILE (BUFDIM.LT.1.OR.BUFDIM.GT.BUFD)
136   WRITE(TERM,('( 'SENTER NUMBER OF WORDS PER OUTPUT RECORD: ')))
137   READ(TERM,*) BUFDIM
138 END DO
139 LINMAX=#
140 DO WHILE (LINMAX.LT.1.OR.LINMAX.GT.MAXLIN)
141   WRITE(TERM,('( 'NUMBER OF RECORDS TO BE OUTPUT = ')))
142   READ(TERM,*) LINMAX
143 END DO
144 C ***** GET OUTPUT FILE NAME AND OPEN *****
145 C
146 C

```



```

147 C
148 CALL RCPAR(2,OTFILE)
149 OREC-MIN(MREC,LINMAX)
150 JRC=BUFDIM*2
151 OPEN(UNIT=OUTFIL,FILE=OTFILE,ACCESS='DIRECT',RECL=JRC,
152 & .MAXREC=OREC,STATUS='NEW',IOSTAT=ISTAT)
153 IF(ISTAT.NE.0) STOP 'OUTPUT FILE OPEN ERROR'
154 C
155 C***** WRITE INPUT PARAMETERS *****
156 C
157 CALL FTIME(ITBUF)
158 WRITE(LPFILE,('(1H,15A2)')) ITBUF
159 WRITE(LPFILE,('(1H,15A2)')) INFILE
160 C
161 WRITE(LPFILE,200) BUFDIM,OREC,IRCL,MREC,JRC
162 200 FORMAT('INPUT PARAMETERS: /
163 * . NUMBER OF WORDS PER OUTPUT RECORD =',16/
164 * . NUMBER OF RECORDS TO BE WRITTEN =',16/
165 * . INPUT RECORD SIZE =',17, 'BYTES' /
166 * . NO. OF INPUT RECORDS =',17/
167 * . OUTPUT RECORD SIZE =',17, 'BYTES')
168 C
169 C***** READ IN PROBABILITY DATA *****
170 C
171 CALL RCPAR(3,DATFIL)
172 OPEN(UNIT=DTFIL,FILE=DATFIL)
173 DO I=1,16
174 READ(DTFIL,*) PROB(I)
175 END DO
176 C
177 C***** GET RECORDS TO PROCESS *****
178 C
179 WRITE(TERM,*)(NUMBER OF RECORDS IN INPUT FILE:'))
180 READ(TERM,*) DERECS
181 MREC-MIN(MREC,DERECS)
182 C
183 C***** BEGIN FILE PROCESSING *****
184 C
185 BUFMAX=IRCL/2
186
187 DO I=1,MREC
188 CT=1
189 PSBOLD=0
190 PREBIT=0
191
192 CALL LGBUF(LBUF,BUFD)
193 READ(UNIT=PELFI,IOSTAT=ISTAT) (BUFFER(P),P=1,BUFMAX)
194 IF(ISTAT.NE.0) THEN
195 WRITE(TERM,('(1H,15A2)')) INPUT READ ERROR, ISTAT=ISTAT
196 STOP
197 END IF
198
199 DO K=1,BUFMAX
200 BITVARD-BUFFER(K)
201 OSTVARD=0

```

```

202 DO L=15,B,-1
203 IF (BTEST(BITWRD,L)) THEN
204   BITVAL=1
205 ELSE
206   BITVAL=B
207 END IF
208 DO Q=1,3
209   NBR(Q)=PREV((Q-1)+(CT-1))
210 END DO
211 NBR(4)=PREBIT
212 PREV(CT-1)=PREBIT
213 FIND=1
214 MUL=16
215 DO P=1,4
216   IF (NBR(P).EQ.1) THEN
217     FIND=FIND+MUL/2
218   END IF
219   MUL=MUL/2
220 END DO
221 IF (BITVAL.EQ.B) THEN
222   IF (PROB(FIND).EQ.1) THEN
223     OBTRD=IBSET(OBTRD,L)
224     PREBIT=1
225   ELSE
226     PREBIT=B
227   END IF
228 ELSE
229   IF (PROB(FIND).EQ.B) THEN
230     OBTRD=IBSET(OBTRD,L)
231     PREBIT=1
232   ELSE
233     PREBIT=B
234   END IF
235 END IF
236 CT=CT+1
237 END DO
238 IF NO BITS IN WORD
239   OBUFF(K)=OBTRD
240   END DO
241   IF NO WORDS IN RECORD
242     PREV(CT-1)=PREBIT
243   C
244   ***** WRITE RECORD TO OUTPUT FILE *****
245   C
246   C
247   CALL LGBUF(LBUF,BUFD)
248   WRITE(UNIT=OUTFIL,IOSTAT=ISTAT) (OBUFF(Q),Q=1,BUFDIM)
249   IF (ISTAT.NE.B) THEN
250     WRITE(TERM, '(OUTPUT READ ERROR, ISTAT= ',14)') ISTAT
251     STOP
252   END IF
253   END DO
254   IF RECORDS IN FILE
255   C
256   C

```

Page 6 RESTORE Opts: 77/LCVI Wed Aug 14, 1985 11:49:47 am
Original Restoration Program /SLDC/SLDC1/RESTORE.FTN

257 C END OF PROGRAM *
258 C *
259 C *
260 *****
261 *****
 END

Module RESTORE No errors Prog: 8619 Blank Common: None
FTN7X 244B/85B3B4 No warnings Save: None Local Ema: None

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date Jan 30 1986

1. REPORT SECURITY CLASSIFICATION Unclassified		10. REPORT NUMBER AD-A160846	
2. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION AVAILABILITY OF REPORT Unlimited	
4. PERFORMING ORGANIZATION REPORT NUMBER		5. MONITORING ORGANIZATION REPORT NUMBER(S) NCS TIB 85-6	
6. NAME OF PERFORMING ORGANIZATION Delta Information Systems, Inc.	8. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION National Communications System	
7b. ADDRESS (City, State, and ZIP Code) Horsham Business Center, Bldg 3 300 Welsh Road, Horsham, PA 19044		7c. ADDRESS (City, State, and ZIP Code) ATTN: NCS-TS Washington, DC 20305-2010	
9. NAME OF FUNDING SPONSORING ORGANIZATION National Communications System	11. OFFICE SYMBOL (If applicable) NCS-TS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Contract DCA100-83-C-0047	
12. ADDRESS (City, State, and ZIP Code) ATTN: NCS-TS Washington, DC 20305-2010		10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 33127K PROJECT NO. TASK NO. WORK UNIT ACCESSION NO.	
13. SECURITY CLASSIFICATION Scan Line Difference Compression Algorithm Simulation Study			
14. PERSONAL AUTHOR(S)			
15. TYPE OF REPORT Final	16. TIME COVERED FROM TO	17. DATE OF REPORT (Year, Month, Day) 85-08-23	18. PAGE COUNT 84
19. SUPPLEMENTARY NOTES			
20. COSAT CODES FIELD GROUP SUB-GROUP		21. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Facsimile, Modread, Compression Algorithm	
22. ABSTRACT (Continue on reverse if necessary and identify by block number) The purpose of this task was to investigate the efficiency and potential operational effectiveness of the Scan Line Difference Compression (SLDC) algorithm presented in Appendix A of the report. Delta Information Systems (DIS) performed a software simulation study of the SLDC algorithm. The software was run using selected CCITT binary standard images as input, and the results were compared with those of the MODREAD II compression algorithm run with the same input data. The MODREAD II algorithm was chosen for the comparison because it is the most effective compression technique currently available. <i>regarding the simulation</i>			
23. DISTRIBUTION AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS		24. ABSTRACT SECURITY CLASSIFICATION	
25a. NAME OF RESPONSIBLE INDIVIDUAL		25b. TELEPHONE (Include Area Code)	25c. OFFICE SYMBOL

END

FILMED

12-85

DTIC